AFRL-RI-RS-TR-2022-160

CENTER OF EXCELLENCE IN RESEARCH AND EDUCATION FOR BIG MILITARY DATA INTELLIGENCE (CREDIT)

PRAIRIE VIEW A&M UNIVERSITY

NOVEMBER 2022

FINAL TECHNICAL REPORT

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE

■ **AIR FORCE MATERIEL COMMAND** ■ **UNITED STATES AIR FORCE** ■ **ROME, NY 13441**

NOTICE AND SIGNATURE PAGE

Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data does not license the holder or any other person or corporation; or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

This report is the result of contracted fundamental research deemed exempt from public affairs security and policy review in accordance with SAF/AQR memorandum dated 10 Dec 08 and AFRL/CA policy clarification memorandum dated 16 Jan 09. This report is available to the general public, including foreign nations. Copies may be obtained from the Defense Technical Information Center (DTIC) (http://www.dtic.mil).

AFRL-RI-RS-TR-2022-160 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

FOR THE CHIEF ENGINEER:

/ S /
LAURENT Y. NJILLA
Work Unit Manager

/ S /
JAMES S. PERRETTA
Deputy Chief,
Information Warfare Division
Information Directorate

This report is published in the interest of scientific and technical information exchange, and its publication does not constitute the Government's approval or disapproval of its ideas or findings.

REPORT DOCUMENTATION PAGE

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED	
		START DATE	END DATE
NOVEMBER 2022	FINAL TECHNICAL REPORT	APRIL 2015	MAY 2022

4. TITLE AND SUBTITLE

CENTER OF EXCELLENCE IN RESEARCH AND EDUCATION FOR BIG MILITARY DATA INTELLIGENCE (CREDIT)

5a. CONTRACT NUMBER	5b. GRANT NUMBER	5c. PROGRAM ELEMENT NUMBER
FA8750-15-2-0119	N/A	OTHER AF
5d. PROJECT NUMBER	**5e. TASK NUMBER**	**5f. WORK UNIT NUMBER**
		R1NR

6. AUTHOR(S)

Lijun Qian

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Prairie View A&M University
100 University Dr,
Prairie View TX 77446

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)	11. SPONSOR/MONITOR'S REPORT NUMBER(S)
Air Force Research Laboratory/RIGA 525 Brooks Road Rome NY 13441-4505	AFRL/RI	AFRL-RI-RS-TR-2022-160

12. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for Public Release; Distribution Unlimited. This report is the result of contracted fundamental research deemed exempt from public affairs security and policy review in accordance with SAF/AQR memorandum dated 10 Dec 08 and AFRL/CA policy clarification memorandum dated 16 Jan 09.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

Sustainable institution to conduct and attract research to which it otherwise would not have access. Increased level of student involvement, papers produced, and impact on the educational program. Demonstrates ability to work within DoD business processes. Desire by Service S&T organizations to initiate their own HBCU-MI programs based on this program's success.

15. SUBJECT TERMS

Center of Excellence in Research and Education for Big Military Data Intelligence (CREDIT), Dempster-Shafer Theory (DST), Texas Higher Education Coordinating Board (THECB)

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES
a. REPORT	b. ABSTRACT	C. THIS PAGE		
U	U	U	SAR	273

19a. NAME OF RESPONSIBLE PERSON	19b. PHONE NUMBER *(Include area code)*
LAURENT Y. NJILLA	**N/A**

PREVIOUS EDITION IS OBSOLETE.

STANDARD FORM 298 (REV. 5/2020)
Prescribed by ANSI Std. Z39.18

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1.0 SUMMARY

The Center of Excellence in Research and Education for Big Military Data Intelligence (CREDIT) has been established at Prairie View A&M University in April 2015 with $5 million funding plus an additional $1 million in 2020 from the Office of the Under Secretary of Defense for Research and Engineering (OUSD(R&E)). The mission of the CREDIT center is to accelerate research and education in predictive analytics for science and engineering to transform our ability to effectively address and solve many complex problems posed by big data, and train our students to become next generation data scientists and engineers.

CREDIT center hosts a multi-disciplinary team of faculty researchers from Electrical and Computer Engineering and Computer Science, research scientists and postdocs, and many graduate and undergraduate research assistants at Prairie View A&M University (PVAMU), an HBCU. The core facilities have been built that include the Deep Learning Lab and the Cloud Computing Lab. The team and computing resources in the CREDIT center allow the team to solve many challenging problems in big data analytics and artificial intelligence and the CREDIT center is leading the curriculum development in big data science and deep learning at PVAMU. CREDIT center has actively collaborated with many academic institutions, government agencies and industry partners to address the challenges in big data analytics and train the workforce for the future data-centric economy. It has played an important role in promoting PVAMU to become an R2: Doctoral Universities - High research activity institution by Carnegie Classification of Institutions of Higher Education recently.

During the funding period of this project, the team of the CREDIT center at Prairie View A&M University and our collaborators at the Stony Brook University and the University of Nevada Reno had completed all proposed research, education, and outreach activities. Significant research results have been obtained. Specifically, (i) forty-five (45) journal papers, two book chapters were published, and (ii) one hundred and four (104) peer-reviewed conference papers (including two best paper award) were published and presented, plus (iii) three (3) journal papers were submitted and under review. The detailed list of publications is given in Appendix A.

The highlights of the technical contributions are summarized as follows: In research thrust 1, a customized domain-specific big data analytics cloud for CREDIT research has been built. The

concept of integrating HPC state-of-the-art technology into big data analytics for performance and scale has been proposed. It has been implemented and tested in a Distributed Volumetric Data Analytics Toolkit on Apache Spark. In research thrust 2, an efficient privacy preserving intelligent edge computing framework has been proposed and implemented. In order to achieve robust data collection and aggregation, computation offloading has been proposed to jointly optimize communications and computing. Then a multitask learning approach has been applied to computation offloading optimization that reduce the inference time by 4-order of magnitude while achieving better accuracy. In research thrust 3, the feasibility of using Dempster-Shafer Theory (DST) and Dezert-Smarandache Theory (DSmT) for big data processing has been explored and a detection framework to mitigate the effect of uncertainty using Evidence Theory (DST - DSmT) and KullbackLeibler (KL) divergence for distance measures is proposed and studied. In the case of limited labeled data, the proposed semi-supervised learning can obtain high inference accuracy using even very limited labeled data, which is a promising solution for real-time machine learning applications. In research thrust 4, a novel multi-task learning based deep learning model has been designed and tested for object identification and target tracking on UAVs. It achieved real-time processing (> 20 fps) and high IoU (> 60%) during real world experiments. Furthermore, a cloud-based big data visualization system is built and achieved real-time data visualization on cloud. The detailed background, literature review, problem formulation, proposed approaches and methods, and experiments and results analysis are provided in chapters 2-4 of this report.

The CREDIT center has maintained close collaborations with program managers and researchers from DOD and has made great effort to contribute to the workforce development for the DOD and the nation. There have been large number of graduate and undergraduate students actively participated the research activities in the CREDIT center, and ten (10) doctoral students and thirty-four (34) masters students and more than a hundred undergraduate students supported by the CREDIT center graduated. All of the graduated students have excellent jobs at government agencies and private industry such as AFRL, NAVSEA, IBM, Intel, Microsoft, Apple, Amazon, HPE, and Dell. All the students conducting research in the CREDIT center have gained deep knowledge and extensive training on AI, machine learning, and big data analytics. They have published significant research results in high quality journals and they have demonstrated outstanding capabilities to solve very challenging real world problems. For example, the CREDIT team including six of our graduate research assistants supervised by the PI (Qian) and Co-PI (Obiomon) participated the AI

Tracks at Sea Challenge organized by the US Navy in Fall 2020. The CREDIT team won the FIRST place out of 31 participating universities including top research universities across the country. In addition to students becoming full time employees in DOD, more than twenty ROTC students have been trained with data analytics skills in the CREDIT center. Furthermore, many students from the CREDIT Center have participated the Summer Intern program offered by the DOD.

The research infrastructure has been greatly improved with the establishment of the CREDIT center. Specifically, the Deep Learning Lab has been developed with 4 NVIDIA DGX-1 systems totaling 32 P100 GPUs with more than 112,000 CUDA cores, plus 4 Dell storage servers with 120TB. The Cloud Computing Lab hosts a high performance computer cluster with 4 racks consisting of 56 IBM dual-core blade servers, 8 HP 16-core nodes, 24 IBM 16-core nodes with GPUs, as well as a 4 Dell nodes for setting up cloud virtual machine farm. These high performance computing facilities provide valuable opportunities for faculty and students to access state-of-the-art equipment to explore cutting edge technologies.

The CREDIT center has been leading the curriculum development at PVAMU to ensure the students receive ample mentoring and training, and the center and PVAMU stay at the forefront of AI and big data education. Specifically, a Deep Learning for Artificial Intelligence Certificate Program had been developed and approved by the Texas A&M University System and the Texas Higher Education Coordinating Board (THECB). The first cohort of fourteen (14) students just received the certificate in 2021. The CREDIT center has also carried out many outreach activities, such as organizing an annual Workshop on Mission-Critical Big Data Analytics, a seminar series, and CREDIT center summer camp for high school students.

Leveraging the research and education capabilities of the CREDIT center, many new grants have been obtained recently built on the momentum of active research and made the center sustainable. With the strong support of the government agencies especially DOD and our academic and industrial partners, the team is confident that the CREDIT center will further improve its research and education capacity and continue to train students especially underrepresented minorities to be highly qualified workforce and contribute to DOD missions and the nation for years to come.

2.0 INTRODUCTION

The research objective of this project is to *amplify the power of predictive data analytics and develop a comprehensive, integrated, and scalable data analysis and inference infrastructure*. The CREDIT center's mission is to accelerate research and education in predictive analytics for science and engineering to transform our ability to effectively address and solve many complex problems posed by big data specifically for military applications.

Today's military intelligence analysts are faced with the monumental and escalating task of handling massive volumes of complex data from multiple sources. This includes sensor data, mobile social network data, surveillance data (such as images and videos from UAVs or satellites), and public domain data. An example scenario is given in Fig.1. The data must be aggregated, evaluated, correlated, and ultimately used to support a commander's time-critical decisions and actions. However, currently there is lack of capability to process huge volume of data from heterogeneous sources in military operations [1, 2]. There exist many challenges such as (1) A real-time computing platform for military big data where massive amounts of data are distributed across locations need to be designed and optimized; (2) Heterogeneous data have to be aggregated in a hostile environment and properly stored; (3) Distributed situational awareness and decision making need to be accomplished with minimum delay; (4) Massive datasets and sophisticated results must be presented for easy perception by analysts.

In order to establish and enhance the ability for more effective and efficient big data processing, a multidisciplinary team of researchers (PI: Lijun Qian, Co-PIs: Lei Huang, John Fuller, Xiangfang Li, Pamela Obiomon, Yonggao Yang) from Prairie View A&M University (PVAMU) collaborating with a team from Stony Brook University and the University of Nevada Reno establish "Center of excellence in Research and Edu- cation for big military Data InTelligence (CREDIT)". The CREDIT center address these fundamental challenges and bring together sensing, perception, and decision support for mission-critical applications of the DOD. Specifically, an integrated computing, communication, and information fusion approach has been proposed and developed. Four

research thrusts have been carried out: (1) System architecture design for a real- time military big data cloud computing system; (2) Secure and robust data

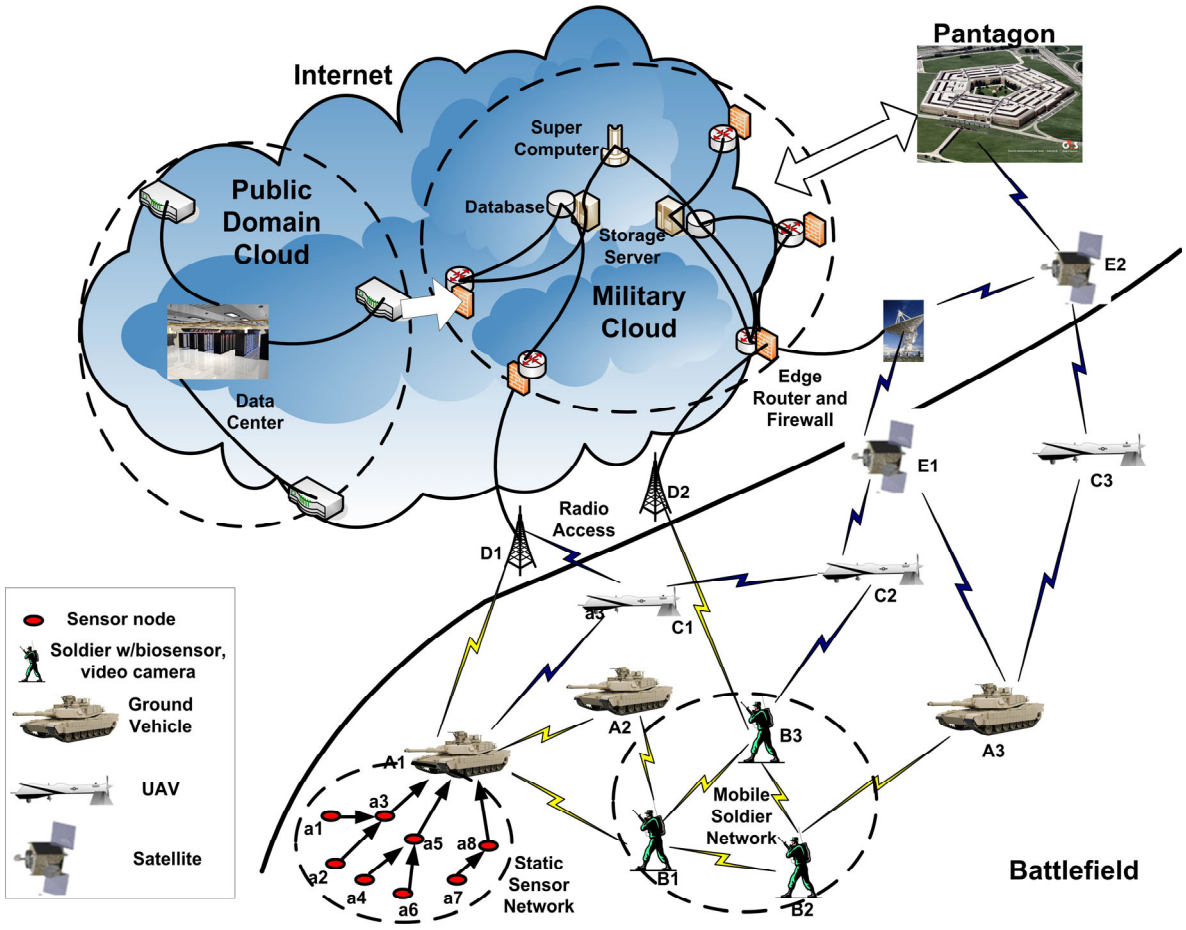


Figure 1: A collaborative information aggregation and processing concept for near-real-time detection and decision making

.

collection and aggregation using edge computing and computation offloading; (3) Novel machine learning and deep learning algorithms for automatic detection using high-dimensional dataset and semi-supervised learning in the case of limited labeled data; (4) Visualization of massive datasets in real-time on cloud and experiments to validate the research results. Together they provide study of relationships among objects and events of interest within a dynamic environment and leverage

data in a particular functional process or application to enable context-specific insight that is actionable. The proposed design and methods have been validated by extensive simulations and experiments using UAV for object detection and tracking as a case study.

In general, majority of today's big data research has been focusing on generating, documenting, organizing, and managing data in public and private repositories [3–5]. However, how to exploit those data is essential for many mission critical applications. The US military has an ever increasing need to obtain intelligence through big data analysis, such as monitoring live video feeds and searching large volumes of archived data for activities of interest. For current operations, more analysts are assigned to watch the same video stream simultaneously. However, analysts are a scarce resource within the military and future datasets are expected to be more sophisticated. Clearly, designing automated search and detection could provide dramatic payoffs in the effectiveness and efficiency of military operations. Because of the predominant effect of time sensitivity of information for war fighters in the theaters with the greatest potential for conflict, the proposed research will focus on time-sensitive data analytics, which analyzing both streaming data such as real-time video streams and archived data such as an image database in an automatic and timely fashion.

In many mission critical applications, the ability to perform analysis on the data is constrained by the increasingly distributed nature of modern data sets. Highly distributed data sources present challenges due to diverse natures of the technical infrastructures, creating challenges in data access, integration, and sharing. Cloud computing is offering an attractive means to acquire computational and data services on an "as needed" basis, which addresses the need for elasticity in many practical scenarios. The distributed nature of data sources also creates additional challenges due to the limitations in moving massive data through channels with limited bandwidth, especially in a hostile environment. Hence, *we propose an integrated design of secure and robust data aggregation and cloud computing based processing for military big data analysis*. Furthermore, challenges exist in better visualizing massive data sets. While there have been advances in visualizing data through various approaches, better methods are required to analyze massive data, particularly data sets that are heterogeneous in nature and may exhibit critical differences in information that are difficult to summarize. Thus, an interactive visualization approach is proposed. In this context, a collaborative information aggregation and processing concept for near-real-time event/anomaly

detection and decision making is proposed in this project (see Fig.1), where a military cloud is overlaid on the Internet and draws data from variety of sources including proprietary data from static sensor systems, surveillance devices on UAVs, input from soldiers' mobile networks, geographic and social network information from public cloud, etc.

As illustrated in Fig.1, when the massive data stream into the military cloud, the cloud needs to dynamically allocate sufficient resources to store and process the data. In order to meet the real-time requirement, we need to explore how to better distribute data and tasks over cloud infrastructure, how to enable elastic computing to meet the dynamic computation workloads, as well as to how to efficiently utilize the additional accelerators to speedup intra-node tasks (research thrust 1).

Given the ever changing environment of battlefield, the data need to be collected reliably and analyzed in decentralized and multi-level fashion. For instance, soldier B3 jammed from communicating to D2 may use cognitive radio to switch channel and report data through C2. Thus, cognitive radio sensor network for robust data aggregation is essential (research thrust 2).

Massive data collected can be redundant or useless that need to be filtered out at an early stage, while some useful data may be lost or missing that need to be derived or collected from other nearby devices. For example, belief propagation may be used among tank A1, A2 and UAV C1 to process their respective collected data, then enemy targets detection can be performed with minimum delay using quickest detection in the military cloud when D1 forwards the updated beliefs from A1, A2, and C1. This proposed approach takes advantage of temporal and spatial correlations of the data and effectively mitigate the effect of missing or incorrect data (research thrust 3). Moreover, we also propose to conduct battlefield specific visualization research in research thrust 4 to intuitively present the massive and complex information in real-time. The theoretical results have been validated through extensive simulations and experiments using test bed. The synergy among the proposed research thrusts is shown in Fig.2. From the information ow perspective, various type of data from different sources will be aggregated in a secure and robust manner and feed the cloud computing system for real-time processing. Machine learning techniques will be applied to perform detection with minimum delay and low false alarm rate. Then proper decision making can be carried out with the help of an interactive visualization tool. Together they provide real-time re-

sponses to critical information needs, accurate knowledge extraction, and risk-aware decision making. As a result, the collaboration among the research thrusts in the proposed architecture fulfills the needs for effective and efficient information sharing and decision making in military operations.

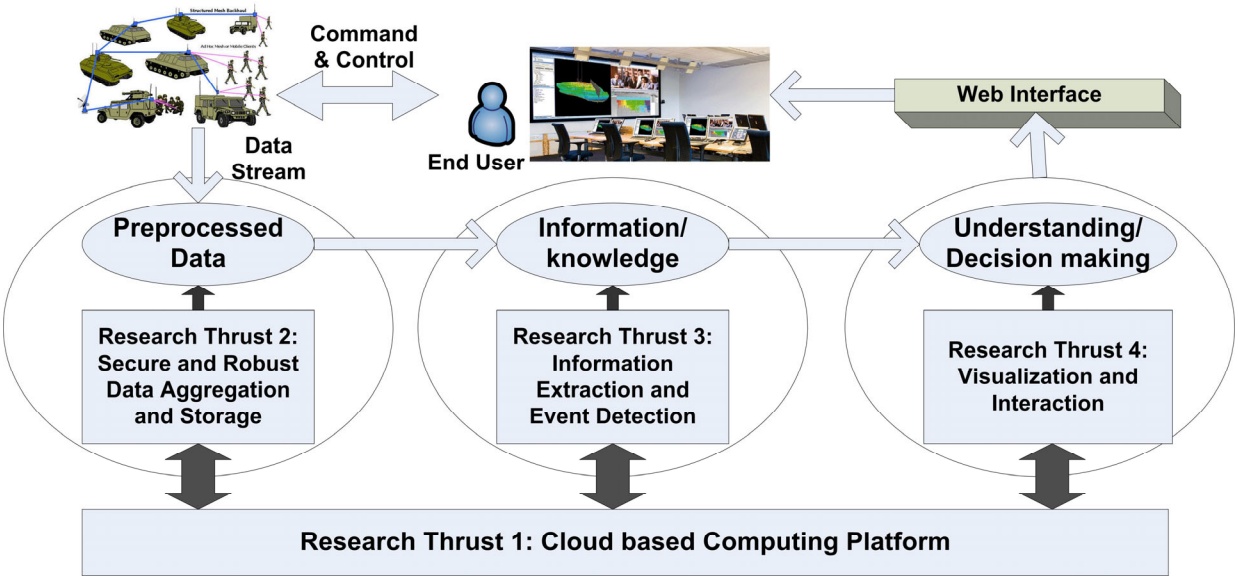


Figure 2: Synergy among the proposed research thrusts

.

3.0 METHODS, ASSUMPTIONS, AND PROCEDURES

3.1 Big Data Cloud Computing System

3.1.1 State-of-The-Art

The big data problem requires a reliable and scalable cluster computing or cloud computing support, which has been a longstanding challenge to scientists and software developers. Traditional High Performance Computing (HPC) research has put significant efforts in parallel programming models including MPI [6], OpenMP [7], and PGAS languages [8{10], compiler parallelization and optimizations, runtime support, performance analysis, auto-tuning, debugging, scheduling, and more. However, these efforts mostly focused on scientific computing, which are computation-intensive, while big data problems have both computation- and data-intensive challenges. Hence, these traditional HPC programming models are not suitable to big data problems anymore. Besides scalable performance, tackling big data problems requires a fault-tolerant framework with high-level programming models, highly scalable I/O or database, and batch, interactive and streaming tasks support for data analytics.

MapReduce [11] is one of the major innovations that created a high-level, fault-tolerant and scalable parallel programming framework to support big data processing. The Hadoop [12] package encloses Hadoop Distributed File System (HDFS), MapReduce parallel processing framework, job scheduling and resource management (YARN), and a list of data query, processing, analysis, and management systems to create a big data processing ecosystem. Hadoop Ecosystem is fast growing to provide an innovative big data framework for big data storage, processing, query, and analysis. However, MapReduce only supports batch processing and relies on HDFS for data distribution and synchronization, which have significant overheads for iterative algorithms. Furthermore, there is no support for streaming and interactive processing in MapReduce, which is the deal breaker for supporting time-sensitive data processing applications.

Storm [13] originally conceived and built by the team at BackType/Twitter to analyze the tweet stream in real time. The goal of Storm is to make it easy to write and scale complex real-time computations on a cluster of computers. Storm guarantees that every message will be processed, and it is able to process millions of tweet messages per second with a small cluster. Storm-YARN enables Storm applications to utilize the computational resources in a Hadoop cluster along with

accessing Hadoop storage resources such as HBase [14] and HDFS. Although it is scalable to process streaming messages, Storm is not designed for batch and interactive execution, and mostly focuses on text-based message processing.

The closest streaming processing engine comparing with Storm is Yahoo S4 [15], and other comparable systems include Esper [16], and Streambase [17]. They are different from Storm in built-in data storage layer, underlying message passing library, and runtime environment. Storm also requires an external database like Cassandra [18] with Storm Topologies to keep persistence. None of the above can support big data processing other than streaming model.

Spark [19] is a quick-rising star in big data processing systems, which combines the batch, interactive and streaming [20] processing models into a single computing engine. It provides a highly scalable, memory-efficient, in-memory computing, real-time streaming-capable big data processing engine for high-volume, high-velocity and high-variety data. Moreover, it supports high-level language Scala that combines both object-oriented programming and functional programming into a single programming model. The innovative designed Resilient Distributed Dataset (RDD) [21] and its parallel operations provide a scalable and extensible internal data structure to enable in-memory computing and fault tolerance. There is a very active, and fast-growing research and industry community that builds their big data analytics projects on top of Spark. However, there is no built-in real-time scheduling in Spark.

Effective software support for emerging hardware is a key requirement to achieve scalable performance. To embrace the heterogeneity of the hardware system in cloud environment, resource allocation and job allocation in a cloud environment need a revisit. Lee [22,23] use Hadoop as the data analytic system and Amazon EC2 as the cloud environment and adopt progress share as the share metric that helps allocate resources to such an environment in a cost-effective manner. The work in [24] adopts Dominant Resource Fairness (DRF) as the resource allocation policy and leaves scheduling to each application. GPUs are used to obtain dramatic performance gains and address the heterogeneous resource demands in data analytics [25]. This work leverages the Adaptive MapReduce scheduler, enables hardware awareness to the scheduler, and monitors tasks in real-time and dynamically co-schedules accelerable and non-accelerable jobs on heterogeneous cluster. Other related works to schedule tasks in a heterogeneous cluster environment include

LATE [26] that speculatively executes tasks; [27] adopts early re-execution of outliers and network-aware placement of tasks; [28] partitions input data to minimize deviation in the task execution time; and [29] proposed a scheduler that overlaps CPU-bound and I/O-bound tasks to improve the overall utilization of the cluster.

High-level programming models are critical to allow big data applications to utilize accelerators for boosting performance. Java and Scala based Spark application can utilize early efforts in porting Java to CUDA such as JCudaMP [30]. In [30], Java is circumvented to focus on SPMD-style parallelism suitable for accelerators by using JaMP [31], which provides OpenMP-like annotations to Java code. Other ways to use CUDA from Java include [32], where Java's native interface is used to call CUDA functions in C, and [33] creates a CUDA-BLAS binding for Java. The libSh [34] assembles a kernel code by building an AST that can be used to generate CUDA or normal threaded code. CuPP [35] creates data structures that can have different representations on hosts and GPUs. Japonica [36] is a Java port of OpenACC.

The stream programming model was designed to represent a program to process stream data with multiple actors that are connected via point-to-point data streams. There are programming languages and systems (e.g. StreamIt [37], Brook [38], SPUR [39], Cg [40], Baker [41], and Spidle [42]) as well as extensions to general purpose languages such as C/C++ [43] and Java [44] that support this programming model. The StreamIt [37] language represents a program as a set of autonomous actors communicating through FIFO data channels. StreamFlex [44] enables stream programming by combining streams with objects through extensions to Java. Researchers have proposed extensions to OpenMP [7] to facilitate the expression of streaming applications by enabling the expression of pipelined computations through the use of a tasking construct [43]. These research, however, have not been adopted by the big stream data computing community.

3.1.1.1 Domain-specific Cloud for Big Data Processing and Analytics

Apache Hadoop (https://hadoop.apache.org) with MapReduce [45] is the widely used open source framework in cloud computing for storing and processing large amount of data in the scalable fashion. There have been many studies around performance of Hadoop on big data analysis. Hadoop with its ecosystem has been successfully deployed in many fields that require to process big data in batch processing. Hadoop File System (HDFS) supports distributed file system with fault

tolerance feature, which provides a large, global-view, distributed file storage using loosely connected computing node disks together. MapReduce as the main parallel programming model provides a simple but typical parallel execution model that works well for applications with map-followed-by-reduce parallel execution pattern.

Apache Spark (http://spark.incubator.apache.org) is the latest parallel computing engine working together with Hadoop that exceeds MapReduce performance via its in-memory computing and high level programming features. Spark is developed using Scala, which is a high-level programming language that supports both functional and object oriented programming. Comparable to DryadLINQ, Spark is equipped with an integrated environment for programming languages. Spark created a unique data structure called Resilient Distributed Datasets (RDDs), which allows Spark application to keep data in memory, while MapReduce relies on HDFS to keep data consistent. RDD supports coarse grained transformation and logging them to provide fault tolerance. In time of losing a partition RDD can re-compute information using named logs to retrieve lost dataset. Based on RDD, Spark supports more parallel execution operations than MapReduce. Defining RDDs via transformations and using them in various operations is the process of programming in Spark. Since transformations are lazy in Spark they won't compute till they are needed. Moreover, Spark supports three high-level programming languages: Scala, Python and Java, while MapReduce only supports Java. Besides batch processing, Spark also supports streaming and interactive programming, which dramatically attracted the interests of many real-time and analytics applications developers. Spark community is very active in development, and Spark is quickly getting popular due to its unique features. The implementation and experiments of this project are built on top of Hadoop and Spark environment.

3.1.1.2 Implementing a Distributed Volumetric Data Analytics Toolkit on Apache Spark

The multidimensional array is a fundamental data structure that has been widely used in scientific computing. Distributed multidimensional array is supported by most High Performance Computing (HPC) programming models either via language features or libraries. HPF, Co-Array Fortran, UPC, X10 and others support distributed multidimensional array to allow users to explicitly specify how to distribute the array on Distributed Memory System while keeping the global address of a multi-dimensional array. Global Array (GA) is a library-based solution that provides users a shared memory view of distributed multi-dimensional array support. SHMEM is another solution

that provides a partitioned global address space for distributed data structures. Significant efforts have been put to simplify the user interface as well as to optimize the performance of distributed multi- dimensional array via compiler and runtime. Hadoop and Spark are currently the most popular open source big data platforms that provide scalable solutions to store, query, process and analyze big data sets. These platforms deliver dynamic, elastic and scalable data storage and analytics solutions to tackle the challenges in the big data era. These platforms allow data scientists to explore massive datasets and extract valuable information with scalable performance. Many technologies advances in statistics, machine learning, NoSQL database, and in-memory computing from both industry and academia continue to stimulate innovations in the data analytics field.

3.1.2 Motivation

Although there are many research works for big data cloud framework, there is, to the best of our knowledge, no complete solution for real-time, high-volume and high-velocity battlefield streaming data analytics. The solution needs a multidisciplinary approach to combine research in big data analytics, high performance computing, real-time processing, machine learning theory, interaction and visualization. Existing big data or HPC research yield significant performance scalability, however, they may not meet the real-time requirements in a battlefield data analytics. Additional breakthrough in the big data analytics platform research is needed to handle massive streaming data in near real-time fashion.

Heterogeneous accelerator architectures, such as NVIDIA GPUs or Intel Many Core architectures, have shown remarkable performance improvement comparing with conventional homogeneous multicore machines. However, one of the critical challenges to fully exploit the hardware computation capabilities is the demand of productive high-level programming models. In this project, our objectives include how to perform accelerator optimizations for big data applications with high-level programming models.

3.1.2.1 Domain-specific Cloud for Big Data Processing and Analytics

The objective is to have a first attempt to explore and demonstrate the scalability and productivity of using the big data and cloud computing techniques for seismic data processing. In order to achieve the goal, a seismic analytics cloud (SAC) combining both big data platform and cloud computing is created to deliver a domain-specific Platform as a Service (PaaS) to support seismic

data storage, processing, analytics and visualization. We have created a variety of seismic processing templates to simplify the programming efforts in implementing scalable seismic data processing algorithms by hiding the complexity of parallelism. The Cloud environment will generate a complete big data application on top of Spark based on user's kernel program and configurations, and deliver the required cloud resources to execute the application.

3.1.2.2 Implementing a Distributed Volumetric Data Analytics Toolkit on Apache Spark

Although multidimensional arrays are mostly used in scientific computing, many big data sets can also be represented using multidimensional arrays, such as 3D/4D volumetric data or images. For example, the medical image segmentation is mostly applied to 3D medical images, and the geological feature classification is applied to seismic volumes. A distributed multidimensional array will serve as a fundamental data structure to support data analytics to these data, especially for scalable machine learning algorithms. Despite the efforts in optimizing multi-dimensional array in HPC platforms, there is a little effort in addressing the multidimensional array performance in any big data analytics platform yet. In this project, we attempt to address the performance of large distributed multidimensional array on a big data analytics platform by applying the HPC experiences and practices. We are also trying to start the discussion of converging the distributed parallel programming practices for both big data analytics applications and HPC applications. Specifically, we present the main functionalities and implementation of a Distributed Multi-dimensional Array Toolkit (DMAT) on top of Hadoop and Spark platform. We then use a set of large 3D seismic data volumes as our experimental data sets and related 3D data computation algorithms to demonstrate the performance scalability. A comprehensive performance study to analyze the performance characteristics in big data platforms has been carried out.

3.1.3 Problem Formulation and Proposed Approach

We propose to build a real-time big stream data analytics cloud system dedicated to store and process battlefield data. Fig.3 depicts the overall architecture of the cloud and its components. The center of the cloud is Spark computation engine that supports streaming, batch and interactive in-memory big data processing. We will extend Spark by adding our designed internal RDD to support a variety of sensor data, image and videos, and their parallel operations. The data collected from a simulated battlefield will be constantly streamed into the Spark streaming engine to be processed in a distributed fashion. Research will be conducted to meet the real-time requirement

by enabling elastic computing and accelerator optimizations for better utilizing cloud resources. Batch and interactive jobs will also be supported to allow historical data exploitation and analysis. Besides existing machine learning, graph computation, and SQL support in Spark, we will extend it to implement advanced battlefield data analysis algorithms based on data fusion (research thrust 3). Information exploited will be saved into scalable and high availability database for future query and visualization. A web-based battlefield analytics interface will be implemented for user-friendly data analysis and visualization.

Batch

- long run jobs for daily reports
- more sophisticated algorithms
- care more for correctness

Streaming

- real-time reporting
- handle all input data with low latency
- care more for low latency than correctness

Interactive

- step-by-step reporting
- balance between latency and correctness
- flexible with users

Spark supports distributed and parallel in-memory computing in three methods: batch, streaming, and interactive. The all-in-one computing engine not only simplifies the programming efforts to develop big data analysis algorithms, but it also offers a huge potential for computing elasticity. Since stream data processing rarely produces a constant and predictable workload, we will study how to make the Spark engine to steal resources from batch and interactive jobs, and allocate resources to stream data processing jobs when necessary.

Spark distributed computing is based on its internal data structure RDD. Currently, Spark supports business data with mostly text formats, which are very different compared with battlefield data.

We will design and extend Spark RDD to support variety of battlefield data, and enable native Spark distributed computing, such as map, filter, reduce, collection, etc. The RDD design needs to consider the typical computation and data access patterns, and define data distribution and access functions.

We will investigate characteristics of battlefield stream data analytics algorithms, and explore how to minimize data communication and balance workload. The collaborator Chapman's team will focus on accelerator-based optimizations for these algorithms. Chapman's team will evaluate NVIDIA GPUs and Intel Xeon Phi accelerators for the core computational algorithms required for this project to improve the performance and meet the real-time response requirement. The research needs to support runtime integration of the accelerators with Spark/Hadoop framework, and the integration of low-level high performance image processing library and C/C++/Fortran OpenMP/OpenACC based algorithm implementation with high-level programming interfaces such as Scala and Java.

- Understanding characteristics of an application that suggest candidacy for stream based parallelism as opposed to exploitation of data and/or task level parallelism
- Provide adequate information to the user to help partition the memory and computation across the hardware contexts
- Minimize the communication latencies between the each computation kernel
- Effective load balancing for increased throughput
- Performance tool support for the stream model
- Processes for the efficient application of the stream based programming model along with other complementary parallel programming models to real world applications

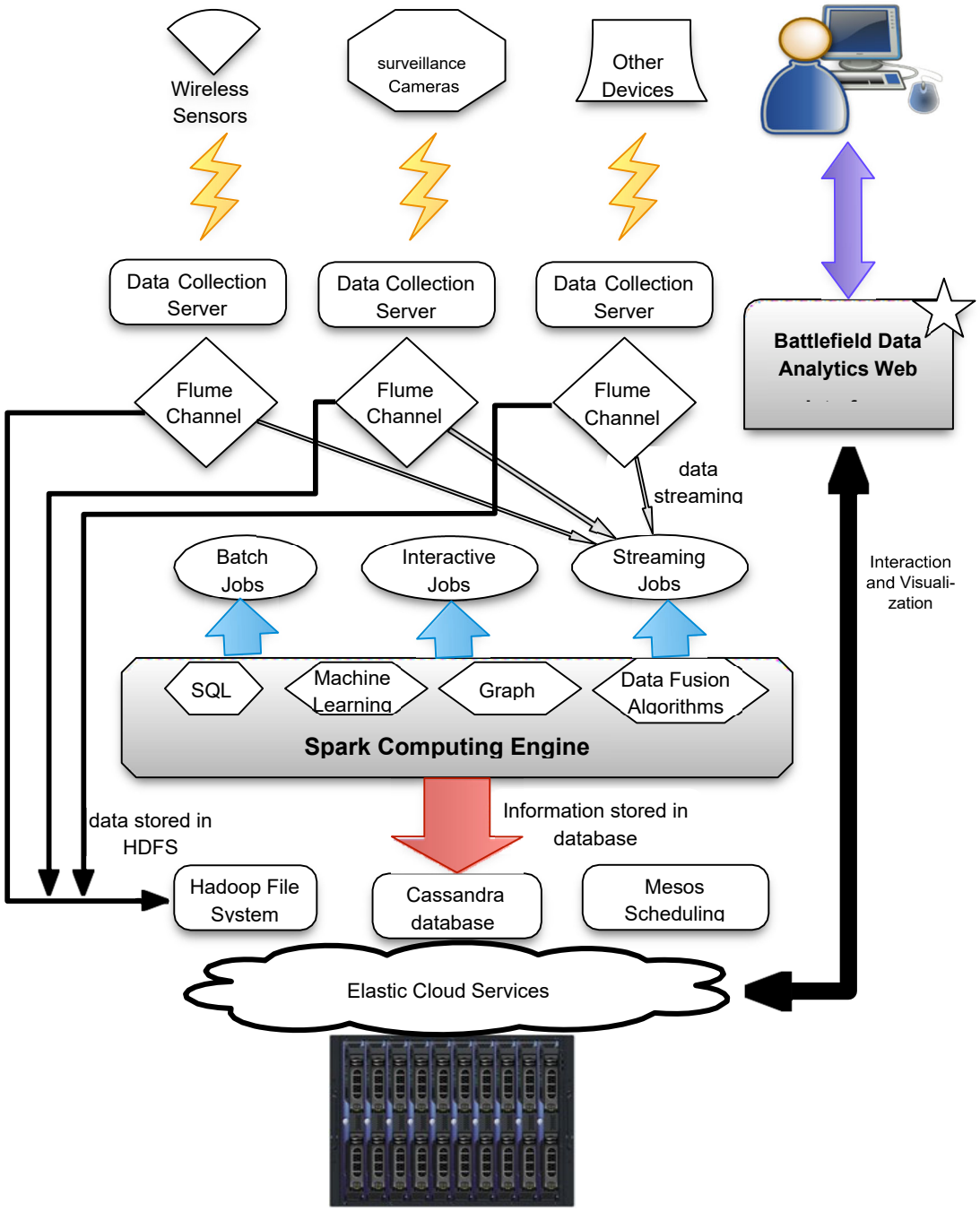


Figure 3: Real-time Battlefield Data Analytics Cloud

The Spark/Hadoop runtime system used by the cloud system schedules mapper and reducer tasks, each of which may invoke core algorithms on CPU or accelerators. The runtime system will be

enhanced to be aware of the performance differences of the same task delivered by different architectures. We will create algorithm-aware and data-aware scheduling heuristics for the runtime system to select the right architecture. Moreover, we will develop Spark streaming task configuration with real-time constraints, and extend its runtime scheduler with real-time scheduling, including clock-driving, priority-driven and weighted round-robin scheduling. We will adopt ideas from BlinkDB [46] to enable real-time data analytics over big data to trade-off accuracy for real-time if necessary.

Spark framework features in-memory computing through the use of aggregated memory space from multiple distributed nodes. However, Spark tasks are not able to directly access other's data even they are physically stored in same node memory. We will explore the potential shared-memory optimizations by integrating the Spark in-memory computation methodology with HPC shared memory runtime systems to explore the possibilities of reducing data movement cost.

3.1.3.1 Domain-specific Cloud for Big Data Processing and Analytics

The Architecture of Seismic Analytics Cloud

The design of SAC architecture is to emphasize twofold: one is to provide a high-level productive programming interface to simplify the programming efforts; the other is to execute users applications with scalable performance. To achieve the first goal, we provide the web interface in which user could manage seismic datasets, programming within a variety of templates, generate complete source codes, compiling and then running the application and monitoring the job running status in SAC. The interface allows users to write seismic data processing algorithms using our extracted common seismic computation templates, which lets users focus on their kernel

algorithm implementation, and simplifies users implementation in handling seismic data distribution and parallelism.

While the most popular-used programming models in seismic data processing include MATLAB, Python, C/C++, Fortran, Java and more, SAC supports Java, Python and Scala natively, so that users can write their own processing algorithms directly on our platform with these three languages; For legacy applications written in other languages, SAC uses the so-called PIPE mode to handle input and output data as standard-in and -out, which requires minor modifications of program source code on handling input and output. SAC will generate complete Spark codes based on users kernel codes and configurations, and then launch and monitor it on the SAC environment.

In order to support large amount data storage and scalable I/O performance, we chose Hadoop HDFS as the underlying file system, which provides fault tolerance with duplicated copies and good I/O throughput by supporting data locality information to applications. HDFS supplies out-of-the-box redundancy, failover capabilities, big data storage and portability. Since the size of seismic data is very large and keeps increasing constantly, HDFS provides a good solution for the data storage with fault tolerance property. We use Spark as the parallel execution engine to start applications, since Spark works well on top of HDFS, Mesos and YARN, and it provides a big data analytics computing framework with both in-memory and fault-tolerance support. Spark provides RDD as a distributed memory abstraction that lets programmers perform in-memory computations on large-scale cluster/cloud in a fault-tolerant manner. To get better performance, we need to put frequently used data into memory and processing data in memory, which is one key performance boost comparing with MapReduce. Some other useful packages and algorithms in data analytics, such as SQL, machine learning and graph processing, are also provided in Spark distribution version. We also integrated some common used libraries for image processing and signal processing, such as OpenCV, Breeze and FFTW etc., to provide a rich third party of libraries support to speed up the development process.

Figure 4 presents the overall architecture of SAC. Based on the SAC web interface, users are able to upload, view and manage their seismic data, which are stored on HDFS. They can then create their application projects by selecting a template from a list of predefined templates to start their own programming. After selected dataset and processing pattern, writing codes and compiling successfully, users can configure the running parameters and then submit jobs to SAC. Job status could be monitored while job is running and running results could be checked after job is finished. On the SAC backend, a big seismic data file will be split into multi-partitions and be constructed into RDD, which will be processed by working threads that apply users algorithm in parallel. After all data are processed, the output data will be saved back to HDFS.

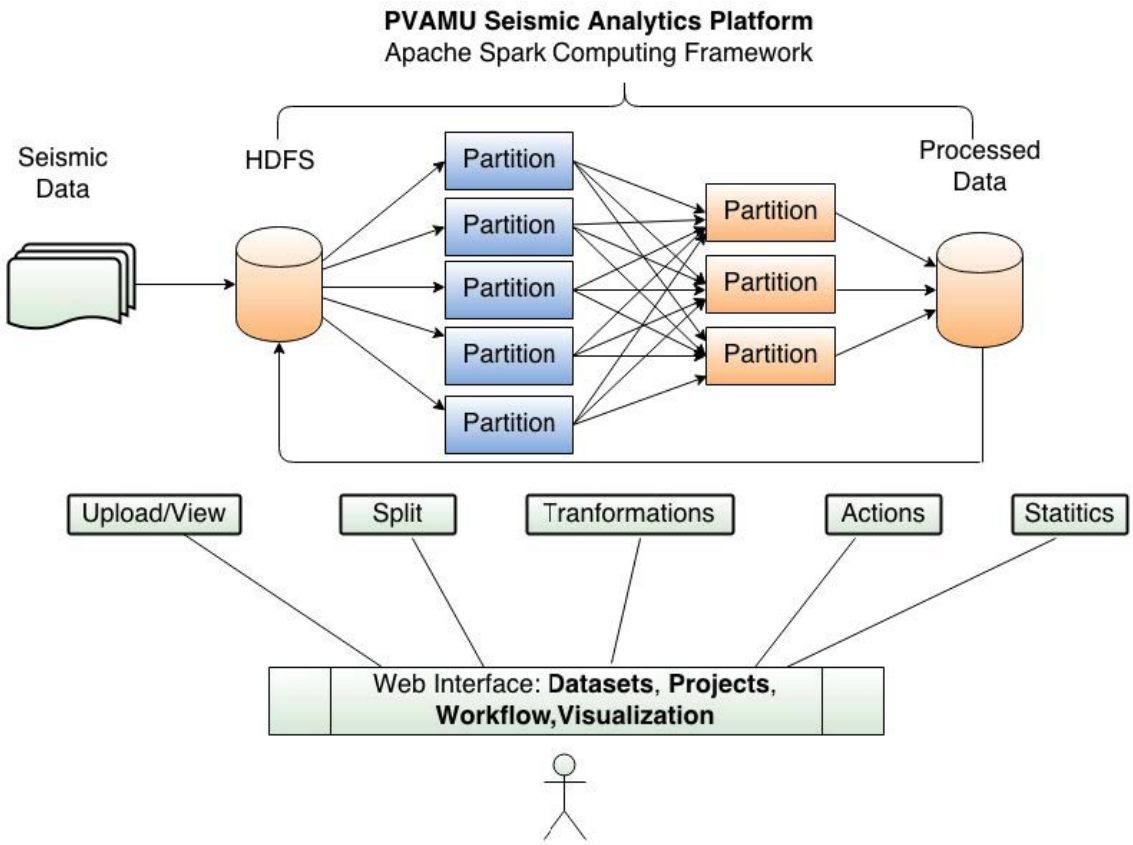


Figure 4: The architecture of seismic analytics cloud platform

Input Data and Redirection

The SEG Y (also SEG-Y) file format is one of several standards developed by the Society of Exploration Geophysicists (SEG) for storing geophysical data. This kind of big seismic data needs to be split into multiple small partitions to be processed in parallel. However, SEG Y data could not be split directly due to its irregularity, so we preprocess the SEG Y data format into a regular 3D volume data, and store the important header in- formation into one xml file. Then the 3D volume data and xml will be feed into Spark applications. Spark uses InputFormat, which is the base class inherited from Hadoop to split such data and construct RDD. Each split will be mapped to one partition in RDD. The embedded InputFormat classes could not handle binary seismic data, so we implemented SeismicInputFormat in this project. Based on configuration defined by user while creating project, such as how many lines each split and number of overlap lines, SeismicInputFormat could spilt the 3D volume and feed partition to each mapper. The data of 3D volume is stored trace by trace in the Inline direction by default. For some algorithms that need to process data in cross-line or time-depth direction, we also provide interfaces to transform Inline format

RDD into cross-line or time-depth direction. In this way, we could cache Inline format RDD in memory, thus all the transformations could be executed in memory with better performance.

Parallel Processing Templates for Seismic Data

Based on the general parallel execution patterns of seismic processing algorithms and applications, we predefined some templates to make this framework easy to program. Every template has explicit input type and output type. The typical templates are: Pixel pattern, which use sub-volume or one pixel as input and output one pixel; Line pattern, which treat one line as input and one line as output; SubVolume pattern, which feed users application with sub-volume and get output from it in sub-volume format. A high level SeismicVolume class has been implemented in this project to provide user interface to access seismic volume. SeismicVolume class provides functions for constructing RDD based on processing templates user had selected, applying users algorithms on RDD, and storing the final RDD on HDFS with format defined by user. To make it easy for programming, we provide some other functions to change the linear array into 2D matrix and 3D volume class; some functional programming interface such as iteration, map/atMap, filter and zip could be used. We also integrated commonly used high-level algorithms, such as histogram, FFT, interpolating and filtering algorithms, so that user could put more attention on data analytics logic instead of details for each algorithm.

Code Generation

After users created project and completed their own kernel codes, one component named Code Generator (CG) in SAC will generate complete Spark codes for running on Spark platform. The CG will parse configuration of users' project and generate Spark application outlined codes, merge them with users' codes. User could also upload existing source codes or libraries, all of which will be integrated into current working project managed by Simple Build Tool (SBT). CG will also generate compiling and running scripts basing on users runtime setting. All these scripts will be called by the web interface, on which some other information such as compiling and running status, location of output will be shown clearly.

Driver and Job Executor

In SAC, every users project will be treated as one Spark application. CG will generate the main driver code for each project. Each application could be submitted to SAC for running after compiled successfully. At execution time, driver code will setup the Spark running time environment, call the SeismicVolume object to generate RDD and execute users algorithms on top of RDD and then store the processed results on HDFS. It will clean up the running environment and release resources after finished. To make it support multiple users, Spark Job- server [15] was introduced to this platform. Based on the priority of application and computation resources requirement of an application, an user could configure the running parameters: number of cores and memory size; and then submit his/her own job, monitoring job status and viewing the running results. Another big advantage of Spark Jobserver is supporting of NamedRDD that allows multiple applications share RDD but has only one copy cached in memory. For some complicate algorithms that need multiple steps or application running in workflow, NamedRDD is a good choice for boosting performance. After job is finished, the running results cloud be discarded or be saved to users workspace basing on users selection.

3.1.3.2 Implementing a Distributed Volumetric Data Analytics Toolkit on Apache Spark

The objective of the work is to develop a scalable Distributed Multidimensional Array Toolkit (DMAT) on big data analytics platforms to enable scalable computation and analytics of volumetric data sets.

Software Stack of DMAT

The software stack of DMAT in a typical application scenario is shown in Figure 5, which simplifies the development efforts for scalable and distributed computing and analytics for volumetric data sets. It is built on top of the Apache Hadoop and Spark. The Hadoop provides a distributed file system (HDFS) and a resource management system (YARN or Mesos), while Spark provides a high-level distributed data representation via Resilient Distributed Dataset (RDD) and a data parallelism execution engine. DMAT provides configurable data distribution fashions for multidimensional data sets, as well as a variety of configurable parallel execution templates to simplify the parallel programming efforts. Moreover, since Hadoop and Spark provide faults tolerance and task scheduling utilities, the toolkit inherits from them to provide fault tolerance and dynamic task scheduling for better reliability and performance.

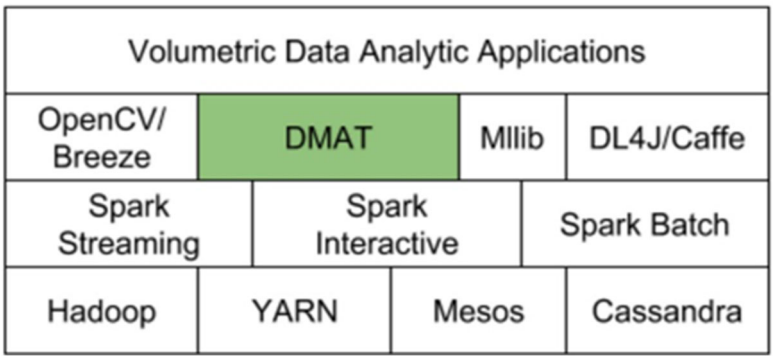


Figure 5: The software stack of DMAT in Big Data Analytics platform

Figure 5 shows the overall big data analytics platform used in our research with the DMAT integrated. The bottom of the figure is the Hadoop Distributed File System (HDFS) that stores the large volumetric data files by utilizing a large number of local disks. The Cassandra as a NoSQL database is also used to store volumetric data, intermediate results, and metadata. MLlib is included in the Spark as the machine learning package to enable machine learning based data analytics algorithms. OpenCV is the widely used image processing package that is used to provide image processing capability. Breeze is the numerical processing package including linear algebra, signal processing, statistics, and other numerical computation and optimizations written in Scala. DL4J (https://deeplearning4j.konduit.ai/) and Caffe (http://caffe.berkeleyvision.org) are the deep learning packages that can be integrated into the Spark platform to apply advanced deep learning technology to volumetric data exploration. We have developed the Volumetric RDD as the distributed multidimensional array to enable parallel operations and machine learning algorithms on Spark.

Data and computational scientists can use DMAT to develop their scalable compu- tation and analytics algorithms by leveraging the capability of Apache Spark, and other packages of image processing, numerical computation, volumetric data interpretation, deep learning, and more. The main components of DMAT are shown in Figure 6, in which the fundamental data structure is the volumetric data RDD building on Spark RDD. There

are two kinds of APIs: one is for the basic operations on the Volume, and the other is high-level template API that makes it easier for a user to develop applications.

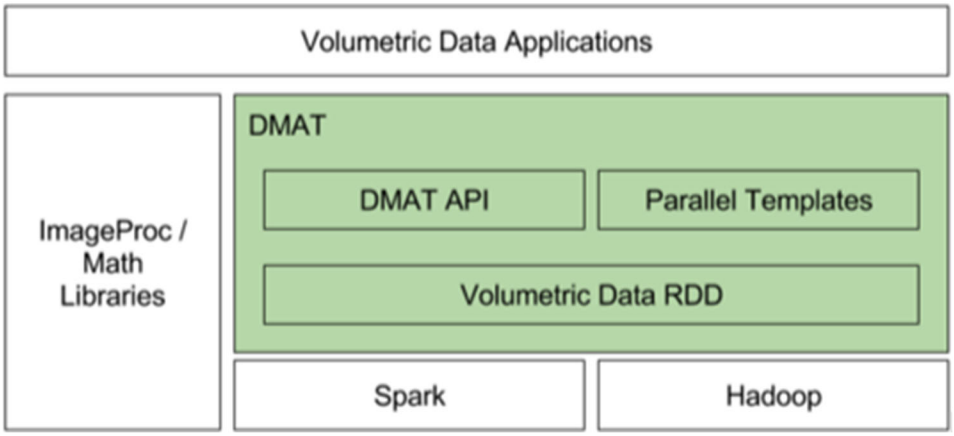


Figure 6: The components of DMAT

DMAT Main Functionalities

Figure 7 shows the main functionalities (APIs) of DMAT, which include volumetric data loading, a variety of data distributions, aggregation, re-partition, sub-volume data accessing and transposing.

DMAT provides a few APIs to load volumetric data from HDFS and to distribute them in any directions (x, y, and z) and granularities to fit users' applications. Distribution with overlapping is implemented to support stencil computation with neighbors.

Moreover, the data aggregation is supported to allow users to re-partition the distributed data during execution. Users can use the APIs to access the distributed data in any directions. 3D volume transposing is supported too. Users can use the APIs and pre-defined parallel templates to apply their own kernel codes running on the Apache Spark in parallel.

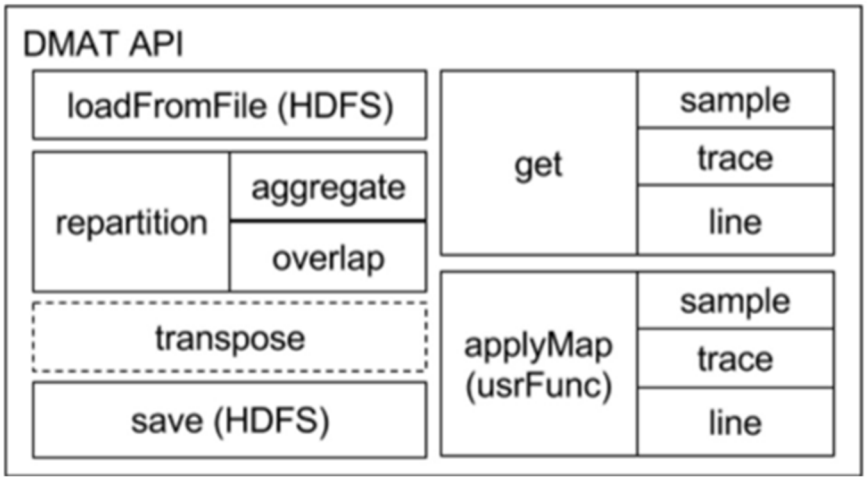


Figure 7: Main Functionalities of DMAT

DMAT loads volumetric data into Apache Spark and creates VolumetricRDD with Float or Double as internal data type. The VolumetricRDD is a derived class from Spark RDD class with a variety of distributed fashions of volumetric data. In addition, it also provides some optional parameters for advanced users who are already familiar with distributed system to specify the advanced data distribution fashions.

Figure 8 shows the ow of distributing a volumetric raw file through the Hadoop file system and Spark RDD, assuming the file has already been uploaded to Hadoop file system. All Spark RDD operations can be applied to the VolumetricRDD. Utilizing the RDD methods provided by Spark, developers could perform various data operations and calculations on the volumetric data in parallel.

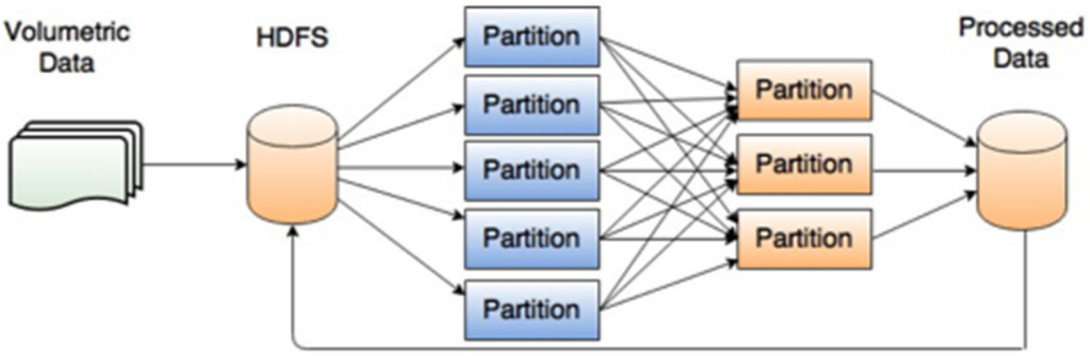


Figure 8: Volumetric Data Distribution Flow

1. Aggregation and Overlapping: Practically, the volumetric data could be viewed as a general 3D volume with Float or Double data in each point. By default, as shown in Figure 9, DMAT distributes the volume in one specific direction slice by slice. In this case, each slice is a single split of the whole data set. Utilizing the RDD operations provided by Spark, we can change the distribution layout to the aggregated and overlapped fashions, as shown in Figure 10 and Figure 11. Users can change the size of distributed splits, and set the overlapped data areas between splits and to access the overlapped parts in each split. Therefore, this method simplifies the stencil-style computation requiring neighbor communication, and make it easy for tuning the performance of distributed tasks, which is a big improvement of the simple map-reduce programming mode.

2. Volumetric Data Access: DMAT allows users to access any slice data in any direction of the volume. Users can specify any index in I, J and K directions, as shown in Figure 12 to access a sub-volume of data. Since Apache Spark does not provide arbitrary data access, the APIs use IndexedRDD to speed up queries of sub-volumes to the master node for visualization or debugging purpose.

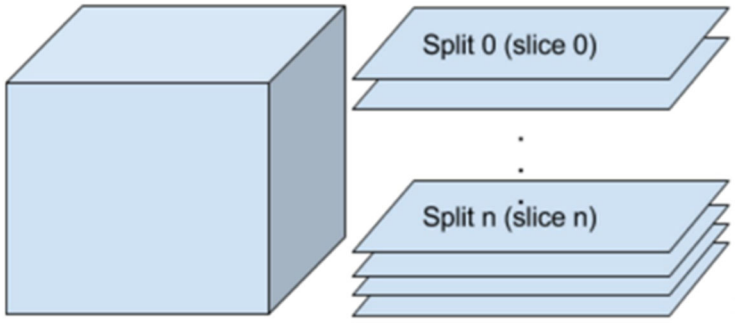


Figure 9: Default distribution schema of Volumetric, planesPerMap=1, overlap=0.

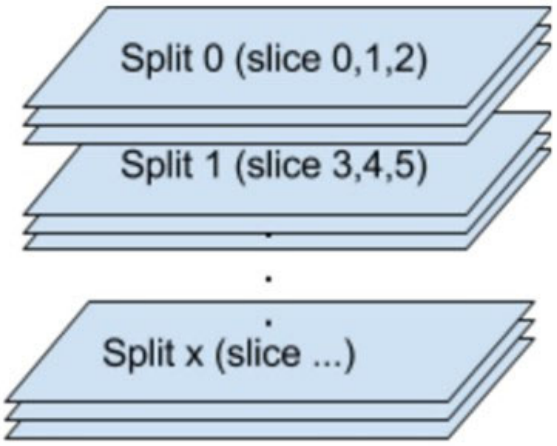


Figure 10: Aggregated distribution of Volumetric, planesPerMap=3, overlap=0

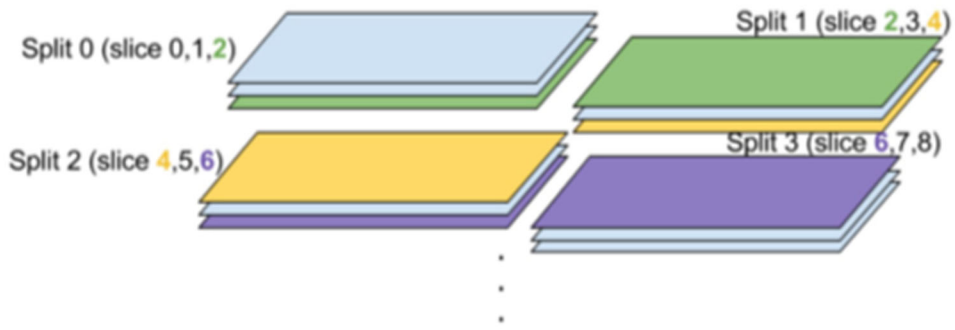


Figure 11: Overlapped distribution of Volumetric, planesPerMap=1, overlap=1.

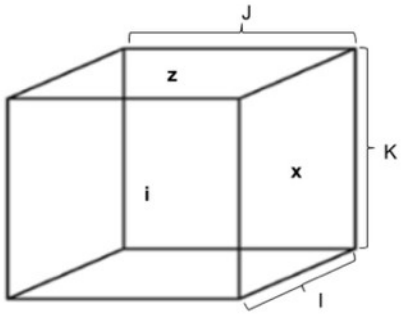


Figure 12: The traditional dimension definition of Volumetric Data

3. Volumetric Data Transposing: Since the volume data could only be stored in file following one specific direction (I, J or K), developers could not access slices of the other two directions directly. To resolve this problem and to achieve reasonable performance, DMAT SDK handles the transposing of the 3D volume data inside the getLine() API and caches all volumetric RDD in three directions.

   To explain the implementation clearly, we denote volumetric data as shown in Figure 12, in which i means I slice, x means J slice and z stands for K slice. The data is stored in i-Slice format. To resolve the transposing problem in each distribution evenly, we split the volume to I of i-Slices in the volumetric RDD and each i-Slice is a 2D matrix. As shown in Figure 3.11, each i-Slice matrix consists of J of i-Traces which have the length of K. An i-Slice matrix could be iterated i-Trace by i-Trace. Since in 3D spacing, each i-Trace is also the trace of x-Slice, for example, the i-Traces(0) is the x-Trace of the 0th x-Slice, the i-Traces(1) is the x-Trace of 1st x-Slice, etc. Thus, we implement a map function to index

all i-Traces of the volume. The new index is combined by index of i- Trace and index of i-Slice. After indexing the map, we got a volume RDD with new (iTraceIndex, iSliceIndex) index as the key, trace data as the value. As we mentioned in Figure 13, to get a x-Slice, we need group all the traces with the same iTraceIndex by utilizing the group operations of Spark RDD. After grouping, we have already got the x-Slices data in our RDD distribution map. To organize them as a x-Slices volume, all we need to do is sorting them by iTraceIndex. So far, the data in requested direction has already been stored in VolumetricRDD, and developers could access any data slice data in any direction efficiently

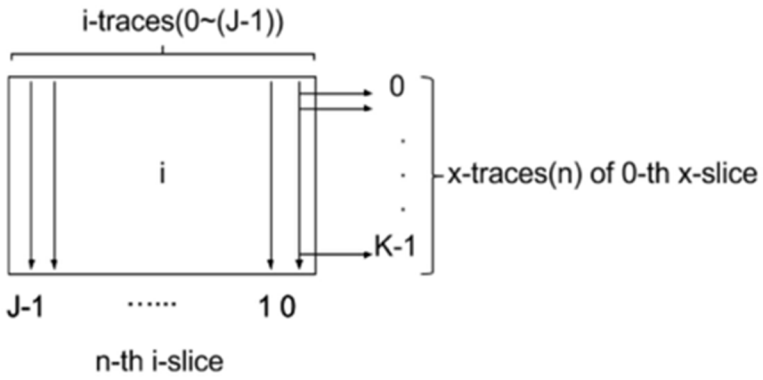


Figure 13: The indexing for resolving transposing problem

User-defined Functions and Parallel Templates

DMAT allows users to define their own functions and apply to the VolumetricRDD in parallel. To avoid involving too much parallelism details when deploying user applications, we also design and implement series of templates for different scenarios of use cases, including apply/iterate by sample, by trace, by line, and by overlapped subvolume. If the user has already been familiar with the distribution system, then the function applying way is recommended since it is more flexible for controlling distribution scheme and tuning performance. Otherwise, the application templates will fit better since they make user deploy their programs on distribution system very quickly without going through all the parallelism details.

1. User-defined function interface: DAMT provides applyMap to allow developers to apply the user-de_ned functions on any direction of the volume in parallel. [VolumetricInstance].applyMap(direction:Int, $f : (T \Longrightarrow U)$). The first parameter direction indicates the direction that users would like to apply the function on; the other parameter $f : (T \Longrightarrow U)$

is a standard spark RDD key-value pairs operation callback function, which feeds the function distributed volume with key-value forms in parallel. The data length in each key-value function depends on the specified distribution parameters of the volume. After execution, this function call will generate a new volume object containing the new data (VolumetricRDD) output by a user-defined function.

2. Parallel Templates: Comparing with traditional sequential codes, the most important thing in parallel programming is data distribution and collection, which provides a big challenge for domain-specific experts. Data distribution also plays key role in distributed parallel programs to achieve scalable performance. In this paper, we implemented several parallel templates to make DMAT be easily used by domain algorithm designers other than computer scientists. Template is actually a kernel function, in which the user only need to take care of input/output on small piece of data, and the toolkit will handle data distribution/collection and parallel computation automatically. These templates defines the data distributions and parallel computation so that users can simply select the right templates for their algorithms without handling the data distribution and parallelism details. Three templates currently include: Line (1D), Plane (2D) and Subvolume (3D). Each template can handle one or more volumes, and will output one or more volumes. Line template is simple, in which the input is a 2D array (dimension 1 for number of volumes and dimension 2 for 1D line data), and output is also a 2D array. Plane template defines a 3D array as input and a 3D array as output respectively. For some computation such as stencil kernel, it not only needs central data, but also requires neighboring data samples, and so in this case, a bigger volume with overlap data will be the input of template. Subvolume template is a good solution to handle data distribution with overlaps, in which both input and output are 4D array. Users can specify parameters about how to distribute data as well as the overlapped areas. In direction K, the whole line will always be put as input. In direction I and direction J, however, user could define the size of center (how much data the computation will affect) and overlap size individually. It is easier to handle the output of template, in which only the valid data without any overlapping will be cached in RDD or persisted to file system.

3.2 Reliable and Robust Data Collection and Aggregation

3.2.1 State-of-The-Art

3.2.1.1 Efficient privacy preserving edge computing for images and video

There is a growing trend of deploying powerful and advanced deep learning models to achieve state-of-the-art performance in processing IoT images and videos. The deployment of the deep learning model either only locally (on IoT edge device) or only at the server fit different scenarios [47]. On the one hand, IoT edge device only training/ deployment is a good choice when the deep learning model is relatively small, and it does not suffer from data privacy and security concerns associated with sending data to the server. However, limited computational power, memory, and energy resource of IoT/mobile edge devices make it di_cult to achieve good latency and energy consumption when training/inferencing on large models [47]. On the other hand, the server only deployment will provide help from edge servers to IoT/mobile devices via computation offloading to handle large models. Although server only deployment achieves scalability, low cost, and satisfactory quality of service (QoS), it suffers communication overhead for uploading the raw data and downloading the outputs, which consume much bandwidth and causes unpredictable latency due to the wireless channel [47, 48]. Also, privacy and security concerns are raised due to the transmission of raw data.

The desire to leverage on the merit of the server only and the IoT/mobile edge device only deployment of deep learning models has necessitated a new paradigm called collaborative intelligence, or collaborative training, or device-edge co-inference [49, 50]. In this new paradigm, the deep learning model is split between the edge device and the server as the computation required for earlier layers is done on the IoT/mobile edge device and the output of the layers called feature tensors are sent to the server for further processing [51]. Despite the advantages of collaborative intelligence in terms of less communication overhead and better data privacy, determining the optimal computation partition point in order to achieve reduced latency and edge device energy consumption is non-trivial because the choice of the best partition point depends on the system factors such as wireless channel state, computation capability of edge devices and edge servers and the deep learning model [48]. Many recent studies proposed various approaches to address this issue, such as [48, 49] Furthermore, it is possible to compress the intermediate features before sending them to the server instead of direct transfer [47, 51, 52].

Several methods are proposed in the literature to address the privacy and security concerns associated with data for training deep learning, such as homomorphic encryption [53], differential privacy [54, 55] and secure multiparty computation [56]. Furthermore, there are many authentication and key agreement schemes that have been proposed to ensure data privacy and security in IoT systems. An authentication framework that uses a digital certificate-based signature scheme that supports efficient signature operations with fast, modular arithmetic operations is proposed in [57]. The authors in [58] proposed ID-based cryptography (IBC) for authentication and the pseudonym-based mechanism for conditional privacy preservation and non-repudiation in urban vehicle communication. A similar authentication method for an edge-based smart grid environment which uses one-way hash functions, XOR computations, and an elliptic curve cryptosystem (ECC), is used in [59]. A framework that uses the cryptography based concept such as physically unclonable functions (PUF) and hash operations to achieve high levels of security at minimal computational resource cost, without requiring storage of security keys is proposed in [60]. Although a similar authentication scheme in [60] is proposed in [61], its uniqueness lies in the use of only one-way secure hash function and bitwise XOR operations for drone and users to authenticate each other. A multi-factor authenticated key establishment scheme based on the PUF, reverse fuzzy extractor, and cryptographic one-way hash function is proposed in [62] for secure smart grid communication.

In addition, some sanitation based methods have also been proposed to ensure data security and privacy. An ant colony system that uses a heuristic function based on pre-large concept and fitness function, with consideration for past selection and current situation, to reduce and monitor the side effects for a designated sanitation procedure is used in [63]. Although a similar ant colony optimization method is proposed in [64], it differs in that it uses multiple objective sanitation models and transaction deletion to hide and secure confidential and sensitive information. The method also uses pre-large conceptual model to reduce multiple scans of the database throughout the evaluation process to achieve lower computational cost. The work in [65] proposed a semantic privacy framework for the Internet of Medical Things, which improves the utility of the sanitized document by identifying negated assertions before the sanitation process and uses industry-standard metrics. Also, many particle swarm optimization (PSO) sanitation frameworks have been proposed. A hierarchical-cluster method, which uses a multi-objective particle swarm optimization framework to hide confidential information, balance the side effects while still discovering useful and meaningful information in the sanitized dataset, is proposed in [66]. The uniqueness of the PSO sanitation

method in [67] lies in the use of the fitness function to minimize the side effects of sanitation by determining the maximum number of transactions to be deleted to efficiently hide sensitive item-sets and pre-large concept to speed up the evolution process. Similarly, PSO method with multiple thresholds and requires minimum support function threshold to hide sensitive information in a utility database is proposed in [68]. It should be noted that the proposed framework is not a substitute for sanitation or authentication methods. Authentication methods are used to establish trust between parties before data transmission to prevent unauthorized access or stealing of data. Data sanitation methods are used to hide confidential information by deleting it. This privacy preserving method is quite different from the aim of this study, where data might not be available to trusted parties due to privacy concerns. Authentication and data sanitation methods can still be used in conjunction with our proposed framework to provide an extra layer of privacy and security.

Despite the success of these methods, some issues remain, such as performance degradation, non-trivial overhead, or limited application [69{71]. The use of collaborative deep learning method, such as federated learning and SplitNN, in distributed learning, has been introduced in recent years to solve the problem of data privacy. Federated learning is a type of machine learning where the goal is to train a high-quality centralized model while the data remains distributed over a large number of clients [72]. It involves sharing model parameters and model gradients through a parameter server without sharing their local data. Federated learning is based on an iterative model averaging, and it is robust to unbalanced data and non-i.i.d. data distribution. Federated learning has been applied to mobile keyboard prediction, vocabulary word learning, and google keyboard query suggestions improvement [73{75]. Federated learning may be viewed as an extension of the idea discussed in [76, 77] that stochastic gradient descent can be implemented in parallel and asynchronously. Federated learning may suffer from non-trivial communication cost. To deal with the high communication cost, an efficient multi-objective evolutionary algorithm, based on a scalable network connectivity encoding method, is proposed in [78]. The use of structured and sketched updates are introduced in [79] to help reduce the uplink communication bottleneck.

Federated learning may also suffer from security/privacy issues due to the need to communicate the model parameters to the central server. One recent study showed potential security/privacy issues due to the possibility of reconstructing original data from the shared gradient [80]. Secure aggregation, a type of secure multi-party computation algorithm for federated learning, is introduced in [81]. Secure aggregation helps guarantee the privacy of data used in generating gradients

shared by each model and improving communication efficiency. Furthermore, it is observed that federated learning performs poorly when the data distributed across the training center is strictly non-i.i.d. of a single class. This statistical challenge is resolved by creating and using a small subset of globally shared data between all the edge devices [82] or adopting a multi-task learning approach [83].

SplitNN can be considered to be a form of collaborative intelligence in a distributed learning environment [84, 85]. The edge device trains the first sub-network up to the cut layer and sends the intermediate features to the server, and the server processes the second sub-network using the received features, a process known as forward propagation. In turn, the server generates the gradient for the final layer, back-propagates the error up to the cut-layer, and sends the relevant gradients to the edge device. The edge device then uses the received gradient to generate the required gradient needed to update the weight [86]. In cases where there is more than one client, the training of the model is done sequentially, which is different from federated learning where the training is done in parallel [86].

The first work on SplitNN is done in [85] where its performance is compared with large SGD and federated learning. It established that SplitNN achieves a significantly lower computational burden on clients and lower communication cost during training than other distributed learning methods. Furthermore, it also showed that SplitNN achieves faster convergence than federated learning when there are many clients. The work on SplitNN in [85] is extended in [87] to use all of the partial clients-networks on each iteration sequentially. This method is suited for vertically partitioned data. It is achieved with each client computes a fixed portion of the computation graph and passes it to the server. The server computes the rest and performs back-propagation, and returns back the Jacobians to the client. Then the client can perform their respective back-propagation. The use of SplitNN to demonstrate the importance of collaborative training of deep learning model using health data is demonstrated in [88, 89]. In [88] several novel configurations of SplitNN are introduced. It also established that SplitNN achieves higher accuracies than that of other distributed learning methods on classification tasks and drastically lowers computational requirements on the client's side. Furthermore, SplitNN requires lower communication bandwidth than federated learning when there are a more significant number of clients. A comparison between collaborative and non-collaborative training modes is carried out, and the impact of the number of clients on the performance of both modes is investigated in [89]. The privacy property of SplitNN is enhanced

in [90] by minimizing the distance correlation between the intermediate features and the input data to reduce leakage. The empirically evaluation and comparison of both federated learning and SplitNN using imbalanced data and non-independent and identically distributed (non- IID) data using real-world IoT settings for performance and overhead (training time, communication overhead, power consumption, and memory usage) is shown in [91]. To leverage on the advantages of federated learning and SplitNN, SplitFed (SFL), a combination of both approaches to eliminate their inherent drawbacks, is introduced in [86].

Compared to collaborative intelligence, where the size of the intermediate feature tensor at the cut layer or optimal layer might be bigger than the original input features, the size of the encoder's intermediate feature tensor in the proposed framework is always smaller than its original input. The encoder compresses the original data to latent vectors, and the compression ratio can be controlled to provide additional flexibility. Uploading of compressed intermediate features to the server leads to lower transmission latency and energy consumption [48, 50]. Although it is possible to compress the intermediate features before sending them to the server in collaborative intelligence, an additional compression step must occur. On the contrary, the proposed autoencoder will extract salient features as latent vectors and perform controlled compression at the same time without the need for additional steps for compression.

In SplitNN, the training of the model is done sequentially. As a result, the training may be very time consuming when the number of edge devices is significant. On the contrary, the autoencoders at the edge devices can be trained in parallel in the proposed framework. Constant communication is also required to exchange intermediate features and gradients in SplitNN, which incurs much overhead and high communication cost. In the proposed framework, the edge device will send the latent vectors only once for the server to train the CNN classifier.

Autoencoder has been applied to address data privacy concerns in several recent works [92{94]. In [92], a convolutional autoencoder that perturbs an input face image to impart privacy to a subject is proposed. It is shown the method can protect gender privacy of face images. A proof-of-concept study has been performed in [94] to use an autoencoder for preserving video privacy, especially when non-healthcare professionals are involved. A modified sparse denoising autoencoder has been applied in [93] to reduce the sparsity and denoise the data. A 3-class classification is performed on the reconstructed data obtained from the autoencoder, and it is shown that the classifier can classify the original black class data as the transformed gray class data. Although autoencoder

has been used to address data privacy concerns, this work is the first to use autoencoder for addressing privacy concerns, communication cost, and deep learning efficiency associated with mobile edge computing systems with a large number of edge devices. The enhanced privacy is achieved using the autoencoder to extract human unintelligible but machine intelligible features from the data. The features or latent vectors are then used to train the classifier. Furthermore, the proposed approach comes with the added advantage of reducing the dimensionality of data needed to be transmitted, reducing the communication cost and the number of model parameters, training time, and inference time. This approach does not suffer from leaking gradient problem associated with federated learning [80] or increase in the size of the intermediate features early on in the models as sometimes observed in SplitNN [48].

3.2.1.2 Computation offloading

As the exponential increase of mobile applications with stringent requirements on computational resources, the mobile devices with limited computation and power supplies become a bottleneck to meet the Quality-of-Service (QoS) of the advanced applications, such as interactive video conference with the requirement of ultra-low latency [95]. In light of this development, instead of mobile cloud computing (MCC) that may introduce large communications latency [96], multi-access edge computing (MEC) plays a key role in bringing cloud functionalities to the edge that near the mobile devices [97]. With computation offloading techniques, the resource-constrained mobile users can save power and enrich the users' application experience by offloading computation-intensive jobs to a nearby MEC server (MES) [98].

The MEC paradigm has attracted considerable attention in both academia [99{109] and industry (e.g., European telecommunications standards institute (ETSI) in [110, 111]) over the past several years. The existing literature in [99{109] has studied a number of problems related to joint optimization of computation offloading strategy and computing resources allocation. In [104{109], the authors formulated the joint optimization of computational resources and offloading decision as a MINLP problem and then solved the problem by seeking an optimal or sub-optimal solution using traditional mathematical algorithms. For instance, the authors in [104] considered the Joint Task Offloading and Resource Allocation (JTORA) as a MINLP problem and resolved using quasi-convex/convex optimization techniques and heuristic algorithm. In [105], the authors jointly considered computation offloading, content caching and resource allocation as a MINLP problem,

which was solved by designing an asymmetric search tree and branch and bound (BB) method. In [106], the authors investigated joint radio resource allocation and edge offloading decision optimization in a multi-cell orthogonal frequency-division multiple access (OFDMA) cellular network, and then proposed variable relaxation and majorization minimization (MM) method to obtain a locally optimal solution. In [107], the authors formulated the energy consumption minimization problem as a MINLP problem and proposed a reformulation-linearization-technique-based Branch-and-Bound (RLTBB) method, by which an optimal or only a sub-optimal result can be obtained. In [108], in order to tackle the MINLP problem to minimize the local user's computation latency, the authors first relaxed it into a convex problem, and then proposed a sub-optimal solution based on the optimal solution to the relaxed convex problem. In [109], the authors formulated the offloading decision, channel allocation, computational resource allocation as a MINLP problem in the multi-access and multichannel interference environment, and then designed a sub-optimal algorithm named as a genetic algorithm based computation algorithm (GACA). Besides, the authors in [99] proposed an optimal offloading strategy using convex optimization. In [100], a game-theoretic computation offloading scheme was proposed for MEC in 5G HetNets. However, due to the diversity of mobile devices' profile and time-varying characteristics of the wireless networks, most of the existing approaches need to solve the MINLP problem very frequently to obtain the optimal or sub-optimal offloading strategy. However, this introduces a large computational overhead to the MEC system and it can hardly obtain the optimal offloading strategy in real time. To tackle this problem, machine learning based approach is an effective and attractive solution. In [101], an optimal offloading scheme was proposed for intermittently connected fog system using Markov decision process (MDP). In [102], a machine learning based runtime adaptive scheduler was proposed for a mobile offloading framework based on past behavior and current conditions. In [103], an offloading decision problem was formulated as a multilabel classification problem for a single-user single-cell scenario, and a deep supervised learning method is developed to minimize the system overhead. Even though the offloading binary decision making problem can be solved in [101{103], the computational resource allocation problem for the resource-limited MES has not been considered.

Compared to the existing literature, the contributions of this work are summarized and emphasized as below.

- Conventional optimization algorithms usually take a long time to solve the MINLP problem, and often only sub-optimal solutions are obtained. Contrary to these algorithms, we propose a multi-task learning (MTL) based approach solve the MINLP problem in real-time. The proposed approach can also adapt to the varying network conditions and the changing requirements of users' applications. Furthermore, the proposed MTFNN model just needs to be trained offline only one time with the dataset collected by traversing all the possible combinations of features including the users' profile and parameters representing wireless channel conditions.
- In this work, both single-server and multi-server MEC system models are considered. Based on the proposed MTFNN model, we further propose a multi-server offloading algorithm to achieve the optimal offloading strategy in the multi-server MEC system. The effect on the system performance from the inference error rate in the classification problem and the inference bias in the regression problem is analyzed. Testing results show that the proposed MTFNN model outperforms the conventional optimization algorithms significantly in terms of computation time (four orders of magnitude) and inference accuracy (up to two times better).

3.2.2 Motivation

3.2.2.1 Efficient privacy preserving edge computing for images and video

Emerging Technologies such as the Internet of Things (IoT) and 5G networks will add a huge number of devices and new services. As a result, a huge amount of data will be generated in real-time. One of the important data types is image data since many applications such as video surveillance, autonomous driving, etc., involve images and videos. To take advantage of the \big image data", data analytics must be performed to extract knowledge from the data. One way to handle the data would be by uploading all the data from edge devices to the cloud or remote data centers for processing and knowledge extraction [112]. However, as highlighted in Figure 14, several factors may render this practice infeasible:1) The sheer volume of the images may overwhelm an uplink with limited bandwidth; 2) The uplink may not always be available, e.g., when wireless communication is used, there might be downtime due to weather(in the case of mmWave), distance, or jamming; 3) Proprietary images may need encryption which introduces additional delay; 4) The end users may have concerns about the security and privacy of their images; thus they may

not agree to upload raw images that may contain private information. Furthermore, uploading is subject to eavesdropping, interceptions, or other unauthorized access.

A novel efficient privacy-preserving framework for image classification in edge intelligent computing systems is proposed in this paper to address these challenges. Specifically, the large raw data will be processed locally (at the edge) by a pre-trained autoencoder. And instead of uploading the raw image, only the compressed latent vectors that contain critical features learned from the raw image will be uploaded through the access point or hub to the edge server for further processing. The proposed framework is highlighted in Figure 15. The experiments demonstrate that the learning performance of extracting knowledge at the server has minimal degradation when the compression ratio is not significant (e.g., below 16 in our test cases). Furthermore, the raw images can be reconstructed with minimal error at the server using the pre-trained decoder if available and needed. The proposed framework encourages the design and implementation of emerging edge intelligent computing that are both efficient and privacy preserving for 5G, IoT, and other advanced technologies.

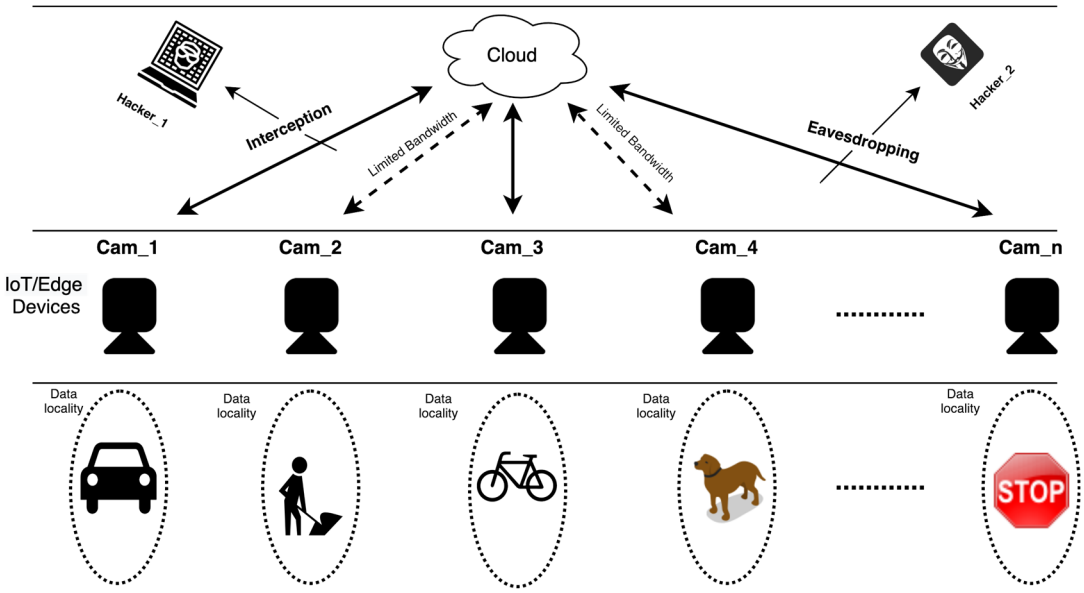


Figure 14: Challenges incurred when uploading all data from edge devices to the cloud

Compared to traditional source coding (e.g., zip), using an autoencoder has the following advantages: 1) Instead of only reducing the redundancy in the raw data as in source coding or traditional data compression, an autoencoder will extract critical features in the raw data and encode the features in a compact form (the latent vector), in other words, the encoder performs initial learning at the edge devices; 2) In addition to compressing the data, the autoencoder also "encrypts" the data by transforming the raw data into latent vectors, thereby enhancing the security of data. For example, a zipped file can easily be unzipped by an adversary if not encrypted; on the contrary, an adversary cannot reconstruct the raw data from the latent vector without knowing the structure (e.g., number of layers, number of nodes in each layer) and all the weights of the pre-trained autoencoder (specifically the decoder part). It is shown in [94] that the autoencoder provides a similar level of security to standard encryption - assuming that the decoder is not shared; (3) Even if an adversary captures the edge device, it is very challenging for the adversary to deduce the decoder part from the encoder part on the edge device.

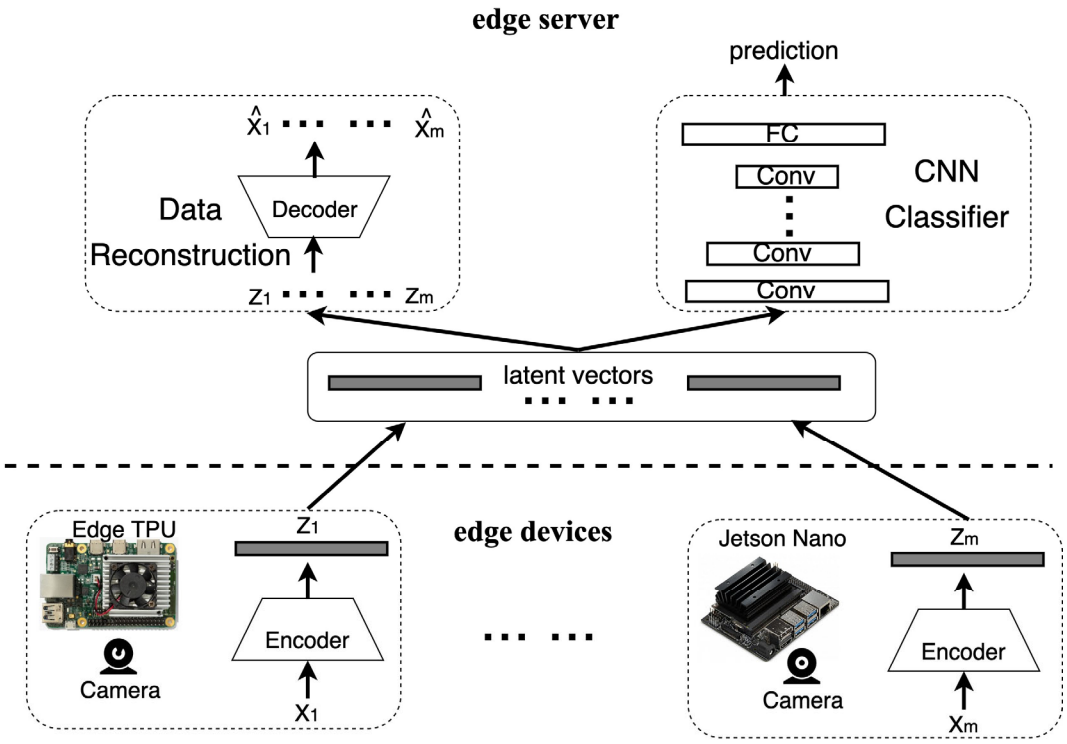


Figure 15: The proposed efficient privacy preserving framework for image classification in-edge computing systems. Here x_i is the raw image, z_i is the compressed latent vector, and $\hat{x}_i$

is the reconstructed image.

The proposed framework has some similar characteristics such as taking advantage of large and diverse data from many edge devices and data locality at each device as in federated learning [72,78,79,113] and collaborative intelligence such as SplitNN [85]. However, compared to federated learning and collaborative intelligence, the proposed framework has the following advantages:

1. In federated learning, the server and the end-users (edge devices) train the same model. As a result, the model's complexity is constrained by the edge device's computing capability and storage capacity. On the contrary, *in the proposed framework, the training of the classifier is done at the edge server only. Thus, it can be deep and complex if needed, and it is not subject to the constraints of the edge devices*.
2. In federated learning, the edge device must rely on the server to update the gradients and train the model. *In the proposed framework, the training of the autoencoder can be done independently at each edge device without any server involvement.*
3. In federated learning, the privacy of the end-users' data is protected by applying differential privacy schemes [114] or through secure aggregation [115], thus introduce additional cost due to encryption or secret sharing. *In the proposed framework, the privacy of the end-users' data is protected by transmitting latent vectors without the additional cost of encryption.*
4. In collaborative intelligence, the volume of the intermediate feature tensor at the early layers or optimal layer might be larger than that of the original input, and so uploading large amount of intermediate features to the server can lead to higher transmission latency and energy consumption [48, 50]. *However, the proposed framework will always compress the original data to latent vectors with the compression ratio controlled to provide additional flexibility. Although it is possible to compress the intermediate features before sending them to the server in collaborative intelligence, this implies an additional compression step is introduced. On the contrary, the proposed autoencoder will extract salient features as latent vectors and perform controlled compression at the same time without the need for an additional step for compression.*

5. In SplitNN, the training of the model is done sequentially. As a result, the training may be very time consuming when the number of edge devices is significant. On the contrary, the autoencoders at the edge devices can be trained in parallel in the proposed framework. Also, constant communication between the edge device and server is required to exchange intermediate features and gradients in SplitNN, which incurs much overhead. *In the proposed framework, the edge device will send the latent vectors only once for the server to train the CNN classifier. This mode of communication leads to low communication cost and overhead reductions.*

3.2.2.2 Computation offloading

In general, the MEC servers are owned by the network operator and could be implemented directly at the access points (APs) or wireless base stations (BSs). To achieve efficient computation offloading in the considered MEC system, there are several challenges need to be addressed.

1. Firstly, to minimize the jobs completion time and energy consumption of the mobile devices, one of the most critical challenges is to determine whether to offload and the portion of computational resources allocated to the offloaded jobs at the MES. Taking into account the computing capability at the device level and communications bandwidth, a joint optimization of offloading decision and computational resources allocation can be formulated as a mixed-integer nonlinear programming (MINLP) problem that minimizes the total system cost (i.e., the weighted sum of delay and energy consumption) under the constraints of the job's tolerable delay and MES's available computational resources. This problem is NP-hard [116], and the optimal solution is difficult and sometimes impractical to obtain in near-real-time for delay-sensitive applications if using the traditional optimization methods.
2. Secondly, the formulated optimization problem should take into account the tradeoff between communications delay and computing time, plus the inherent heterogeneity in terms of mobile devices' computing capabilities, computation jobs' QoS requirements, and the computing resources available at the MES.
3. Thirdly, the input parameters to the optimization problem may change from time to time due to the time-varying network environment and users' applications. This leads to frequent re-computation of the optimal offloading decision and computational resource allocation

in near-real-time. However, conventional mathematical optimization techniques usually converge slowly and have forbidden complexity for real-time implementations.

In order to address these challenges, a novel computation offloading framework is proposed in this work. In order to clarify the fundamental idea and novelty in this work, we summarize the following critical technical aspects.

1) **Offloading Problem Formulation**. First of all, research works on mobile edge computing (MEC) conclude that the offloading problem can be generally formulated as a mix integer nonlinear programming (MINLP) problem, which is non-convex and NPhard. By reviewing the existing literature, the offloading problem can be formulated either (a) as a centralized problem, then the offloading problem is decomposed and solved with relaxation iteration algorithms at the access point (AP) or base station (BS), such as in [117{120]; or (b) as a distributed/decentralized problem, then the offloading problem is reformulated based on game-theoretic approaches and the mobile users/STAs are directly involved to achieve the Nash equilibrium [121]. *In this paper, we adopt the centralized optimization formulation of the offloading problem in MEC and propose an efficient solution using multi-task learning. The distributed optimization formulation is not considered and it is out of the scope of this paper.*

2) **The Motivation**. In practice, some factors (e.g., number of users, the capability of devices, the channel conditions, etc.) may vary over time, so the centralized optimization procedure must be executed repeatedly each time the parameters change. Therefore, the conventional centralized optimization methods using relaxation iteration algorithms incur high computational complexity due to numerical iterations and their solutions are often suboptimal and would not scale well. This is increasingly a concern when 5G and IoT paradigms are taken into account [122]. *To address this issue, one of the better solutions would be utilizing deep neural networks (DNN)-based machine learning (ML) algorithms as explained in below.*

3) **The Proposed Multi-task Learning based Solution**. To solve the MINLP problem more efficiently in centralized offloading, this paper proposes an ML solution, as illustrated in Fig. 16. Specifically, a DNN is designed and trained offline to obtain the mapping from the input parameters such as the number of users and channel conditions to the output (the optimal solution of the offloading strategy of the users). After the offline training, the DNN

can directly infer the solution of the offloading problem given the input parameters. To this end, it is clear that the proposed MTL solution has the following advantages.

- *Generalization*. The data for training the DNN offline can be obtained by performing an exhaustive search of the optimal solutions over a wide range of parameter settings. Then each pair of input parameters and the corresponding optimal solution constitutes a training sample. Although the input parameters during offline training do not include every possible combination of the parameters, they do cover a wide range of parameter settings, and the well-known generalization property of ML models will enable the trained DNN to produce accurate inference of the solution even for parameter settings not included in the training samples. As proved by the probably approximately correct (PAC) learning, generalization can be achieved by designing a proper ML model with enough labeled data [123].

Font = small

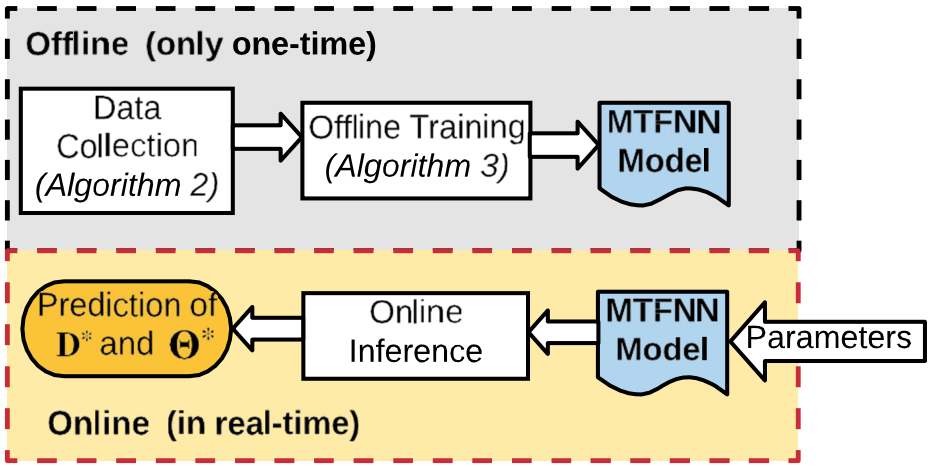


Figure 16: Brief overview of the proposed MTFNN framework

- *Low Computational Complexity yet High Accuracy*. During offloading, the optimal solutions must be obtained in near real-time and repeatedly since the input parameters (such as wireless channels, number of users, traffic payload size and so on) may vary quickly. However, considering that the offloading optimization problems are NP-Hard, the optimal solutions are often very difficult to obtain and it is impractical to use conventional methods such as the Lagrangian relaxation-based algorithms due to their high computational com-

plexity. To address these challenges, the proposed ML-based method \moves" the complexity of online computation to offline training. With the offline-trained ML model, the AP can infer the offloading strategy directly and efficiently with relatively high accuracy by performing feedforward calculation (no iteration is required).

Notations: As per the traditional notation, a bold letter indicates a vector. An upper case letter indicates a random variable or random parameter and a lower case letter indicates a realization of a random variable or random parameter. $min\{.\}$ represents the minimum value and $|.|$ denotes the absolute value. The symbol $\Longrightarrow$ denotes “implies” relation. For ease of reference, we list the key notations in Table 1. In this work, instead of optimization using traditional numerical methods that require many iterations, a neural network based optimizer is proposed to give the offloading solution to the optimization problem by performing the feedforward inference.

In this work, instead of optimization using traditional numerical methods that requiremany iterations, a neural network based optimizer is proposed to give the offloading solution to the optimization problem by performing the feedforward inference.

Specifically, a multi-task learning (MTL) based feedforward neural network (MTFNN) model is designed to solve the MINLP problem in near-real-time. In the proposed MTFNN framework, the offloading decision making is formulated as a multiclass classification problem and the computational resource allocation is formulated as a regression problem. To the best of our knowledge, this is the first attempt to jointly optimize the jobs offloading decision and computational resource allocation using multi-task learning for the MEC system. Although offloading in edge computing is well studied, and multi-task learning (MTL) is also well known, applying MTL to solve offloading strategy optimization problem with high accuracy in real-time is a novel idea.

Table 1: List of Key Notations

Notation	Description
N	Total number of MUs
K	Total number of CAPs
$\mathcal{N}$	Set of all MUs
$\mathcal{K}$	Set of all CAPs
J_n	Jobs of the MU n
s_n	Size of J_n
c_n	CPU cycles required to process J_n
ϑ_n	Maximum tolerable delay of J_n
$r_{(n,k)}$	Uplink data rate between the MU n and CAP k
$D_{n,k}$	Offloading decision of the MU n to CAP k
$f_{n,k}$	Computational resource allocated to MU n by CAP k
f_n	CPU cycle frequency of MU n
F_k	Total computational resources of CAP k
α	Weight of the delay
P^i	Transmission power of MU n to CAP k via sub-band i
h^i	Channel gain for MU n to CAP k via sub-band i
B	Total frequency band
$\mathcal{B}$	Set of sub-band
τ^n	Local execution delay of J_n
ε^n	Energy consumption processing J_n locally
SINR^i	SINR of MU n served by CAP k on sub-band i
$T^{n,k}$	Time to upload J_n to CAP k
$T^{n,k}$	Time to execute J_n by CAP k
$T^{n,k}$	Time to transmit the execution result to MU n
e^n	Consumed energy of MU n during the data uploading
e^n	Consumed energy of MU n during the CAP processing
e^n	Consumed energy of MU n during downloading results
τ^n	Execution delay of J_n using offloading
ε^n	Energy consumption processing J_n using offloading
O^n	Weighted-cost for computing J_n locally
O^n	Weighted-cost for computing J_n using offloading
O_{total}	Weighted-sum cost of all MUs
$(\mathbf{D}_n^*)$ $\mathbf{D}_n$	(Optimal) offloading decision vector for MU n
$(\mathbf{F}_n^*)$ $\mathbf{F}_n$	(Optimal) resource allocation vector for MU n
$(\mathbf{\Theta}_n^*)$ $\mathbf{\Theta}_n$	(Optimal) resource allocation ratio vector for MU n
(S_n^*) S_n	(Optimal) offloading strategy for MU n
S	Offloading strategy for all MUs
S^*	Optimal offloading strategy for all MUs

3.2.3 Problem Formulation and Proposed Approach

3.2.3.1 Efficient privacy preserving edge computing for images and video

Proposed Framework: The proposed efficient privacy-preserving framework for image classification in edge intelligent computing systems is shown in Figure 15. It has two levels: the edge devices and the edge server. It is assumed that the nodes of the edge devices contain sensors such as cameras and embedded computing devices such as Google edge TPU [124] or NVIDIA Jetson Nano [125]. The edge server is assumed to have large storage and strong computational capacity. The edge segment of the framework mainly contains the various edge devices of interest and the pre-trained encoder. The server mainly contains the hub, the pre-trained classifier, and the pre-trained decoder. We only consider supervised learning in this paper, and it is assumed that the training dataset is labeled. One of the motivations for this proposed framework stems from the guaranteed reduction in feature size of the original input when observed at the encoder output. With a feature (latent vector) size smaller than the size of the original input being sent to the server, the latency and the energy overhead and communication cost can be reduced. Furthermore, this provides improved data privacy and security, compared to sending the raw data to the server.

The data from each edge device is passed to the corresponding encoder attached to it. A unique pre-trained encoder is used at each edge device to take advantage of data locality at each device. The function of the pre-trained encoder in the inference mode is to extract the most important and critical features in the data. The encoder also ensures dimension reduction of the input data by a pre-determined factor. The extracted critical features (latent vectors, intermediate features, or feature maps when the data are images) are then transmitted to the hub at the server. The two primary tasks at the server are the classification task and the data reconstruction task (recover a copy of the original image from the latent vectors). In other words, at the server, the latent vectors are input to the pre-trained classifier for prediction and are also input to the corresponding decoder for the reconstruction of the images.

The design of the proposed framework has two (2) stages: the training stage and the testing stage.

Training Stage: The dataset collected at each edge device is used to train an autoencoder for the corresponding device as a way to take advantage of the data locality at each device. Autoencoders are generative models where an artificial neural network is trained to reconstruct its input in an unsupervised way. Figure 17 illustrates all the components of an autoencoder and the training process. It is made up of two main blocks, which are the encoder and the decoder [126, 127]. The encoder compresses the input X into a low dimensional representation of pre-determined size, called the latent vector denoted by Z that contains the most important features in the data. When the input data are images, Z will be the corresponding feature maps. The mapping function of an encoder is stated in equation 3.1 where Z is the encoder output, W is the model weight and be is the bias of the encoder, X is the model input and $f(.)$ is the non-linear activation function.

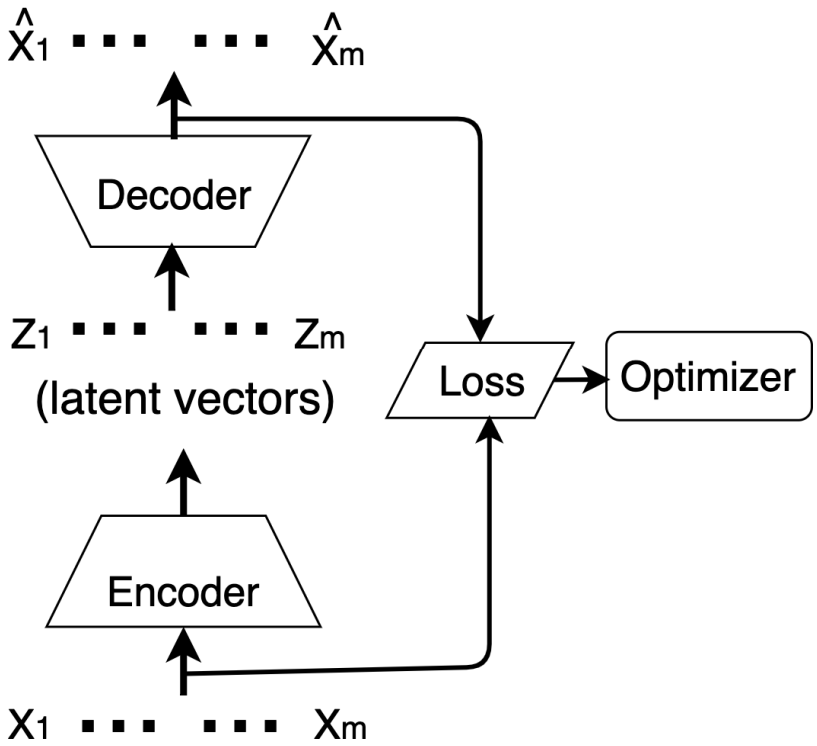


Figure 17: The training for the proposed autoencoder at edge device

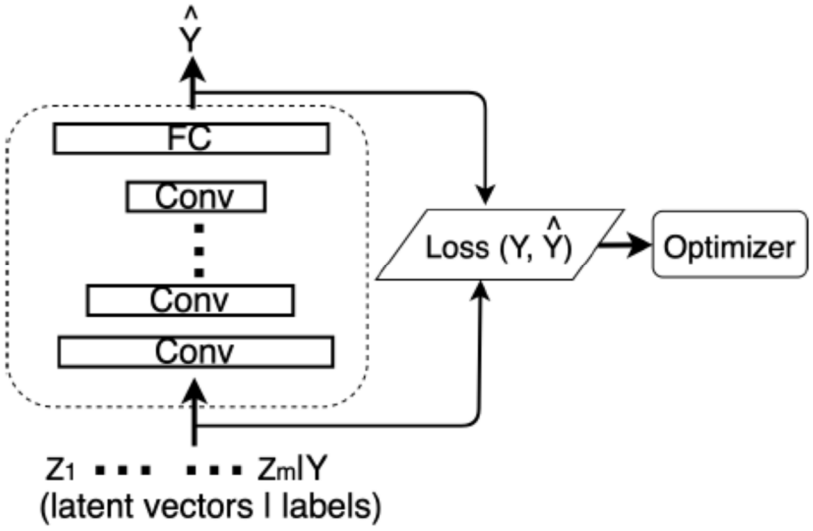


Figure 18: The training for the proposed CNN classifier at the server

$$Z = f_\theta(X) = f(WX + b_e) \tag{3.1}$$

The decoder then tries to reconstruct the original input data/image from the latent vector Z. The reconstructed input data obtained at the decoder output is denoted by $\hat{X}$. It should be noted that an autoencoder is a lossy network as the original image will not be fully recovered. However, it is expected that the critical features will remain in the recovered image. The decoder is represented mathematically in equation 3.2 where $\hat{X}$ is the decoder output or estimated input, V is the decoder weight, Z is the encoder output, b_d is the decoder bias and $g(.)$ is the activation function of the decoder.

$$\hat{X} = g_{\theta'}(Z) = g(VZ + b_d) \tag{3.2}$$

The autoencoder achieves the proper training of the encoder and decoder by minimizing the differences between the original input (X) and the reconstructed input ($\hat{X}$). The training is achieved

using the mean square error (MSE) loss function or any other appropriate loss function. The formulae for the MSE loss function is stated in equation 3.3. After the training of the autoencoder, the encoder part of the autoencoder is then extracted, deployed in the inference mode on the edge device, and then used to generate the latent vector Z. Hence, the dataset is transformed from [X; Y] to [Z; Y] where Y are the labels.

$$L_{\theta,\theta'} = \frac{\sum_{i=1}^{N} \left\| X_i - \hat{X}_i \right\|_2^2}{N} \tag{3.2}$$

The latent vectors and the corresponding labels are aggregated at the hub, and then used to train a classifier on the cloud in a supervised manner, as shown in Figure 18. The type of classifier at the cloud is determined by the type of supervised task to be done. The most common type of classifier used for image dataset is the convolutional neural network (CNN) and it is used as the classifier in this work. CNN is a type of multilayer neural network that preserves spatial relationships by performing convolution operation in order to learn features at different layers. The mathematical equation for a convolution operation is given in equation 3.4 where $S(i;j)$ is called the feature map, $K(i;j)$ is the filter, and $X(m;n)$ is the input. The convolution operation, which is equivalent to an integral that expresses the amount of overlap of K as it is shifted over X, is achieved by taking the dot product of two inputs over a finite number of samples. [127, 128].

$$S(i,j) = (K * X)(i,j) = \sum_{m} \sum_{n} X(i-m, j-n) K(m,n) \tag{3.4}$$

The mathematical representation of forward propagation for a feature map at a particular layer is given by equation 3.5 where S_j^l is the j-th feature map in l-th layer, $S_j^{l-1}(m = 1, \dots, M)$ are the outputs of the $(l-1)$th layer, w_{jm}^l is the weight connected to the m-th feature map in the previous layer, b_j^l is the j-th bias of the l-th layer, and F is the activation function [129]. The cross entropy loss function which results in normalized probabilities is used for training the classifier at the edge server. The mathematical representation of the cross entropy loss is shown in 3.6 where Y_i is the label/ground true and $\hat{Y}_i(0 \geq Yi \geq 1)$ is the prediction probabilities.

$$S_j^l = F\left(\sum_{m=1}^{M} w_{jm}^l * S_m^{l-1} + b_j^l\right) \tag{3.5}$$

$$L\left(Y_i, \hat{Y}_i\right) = -\sum_{i=1}^{M} Y_i \log \hat{Y}_i \tag{3.6}$$

The algorithm for the training phase is stated in Algorithm 1 and Algorithm 2.

Inference Stage: In this stage, the pre-trained encoder, pre-trained decoder, and pre-trained classifier are deployed in the inference mode. The data X from a edge device is fed to the corresponding pre-trained encoder attached to that device. The encoder then transforms the data X to a latent vector Z, representing the most critical feature in X. The latent vector Z, which is smaller than X by a pre-determined ratio, is then transmitted to the edge server. The latent vector Z is fed into the pre-trained classifier at the edge server, and the classifier then predicts a label $\hat{Y}$. The original data is sometimes needed at the edge server in applications such as anomaly detection and security surveillance. When a copy of the original image is needed at the server, the latent vector Z is fed into the input of the corresponding decoder, and the estimate of the original data is obtained. In applications where privacy is very important, the original image might not be requested due to privacy concerns.

Proposed Framework and Data Streaming: The design of the deep learning framework above is done using an offline learning approach. In offline training, historical data is available at the edge to train the autoencoder during the training phase. In IoT data streaming, the proposed framework still applies, particularly in a situation where the streamed data needs to be stored for labeling to take place.

In situations where this is not possible, the proposed framework can still be used with some slight modification depending on the velocity of the data. Firstly, the online learning approach is used as the datasets are not available at once. This means that the traditional backpropagation method used above might change to a backpropagation method that is suitable for online learning, such as hedge backpropagation [130] or use the traditional backpropagation with a batch size of one which is inefficient [131]. Furthermore, there will be a need for a distributed streaming processing platform

such as Apache Flink, Apache Spark, Kafka streams to handle issues peculiar to data streaming such as out-of-order datasets and data buffering in order to balance event-processing with low latency and high throughput [132].

Algorithm 1 Autoencoder model development for image dataset at each edge device

Input: Training Image data at each edge device X. The corresponding labels is also X

Split dataset into training image dataset (**70**%) and testing image dataset (**30**%)

Normalize the data $X = \frac{x-min(x)}{max(x)-min\,(x)}$

Train the autoencoder model:

1: initialize $\boldsymbol{\theta}$

Training Process

2: **for** *numberofepochs* **do**

3: Autoencoder forward pass $\longrightarrow \hat{X}_i$

4: calculate loss function $L \longrightarrow |X - \hat{X}_i|$

5: perform back propagation $\longrightarrow \frac{\partial L}{\partial \theta_i}$

6: update autoencoder weights, $\theta_{i+1} \longrightarrow \theta_i - \eta \frac{\partial L}{\partial \theta_i}$

7: **end for**

8: return $\boldsymbol{\theta}$

Test the autoencoder model: *Testing Process*

9: **for** *X in TestingImagedataset* **do**

10: autoencoder forward pass $\longrightarrow \hat{X}_i$

11: encoder forward pass $\longrightarrow Z_i$

12: Loss $\longrightarrow \frac{\sum_{i=1}^{N} \|X_i - \hat{X}_i\|}{N}$

13: **end for**

14: **return** $Loss, Z$

Table 2: Information on the CIFAR10 and ImageNet (IMGNETA and IMGNETB) Datasets

Dataset	Image size	#of images	Training Testing ratio	#of classes	Comments
CIFAR10	32*32*3	60 000	5:1	10	
IMGNET-A	224*224*3	13,000	7:3	10	very different images
IMGNET-B	224*224*3	13,000	7:3	10	very similar images

Table 3: The deep learning models and the dataset used in training the models

		CIFAR10	IMGNET-A	IMGNET-B
2*Vanilla Model	Model-A	x	-	-
	Model-B	-	x	x
Transfer Model	Model-C	-	x	x

Algorithm 2 CNN Model development for latent variables at the edge server

Input: The compressed/machine intelligible image dataset Z and corresponding labels Y at the cloud

Split dataset into training image dataset (**70**%) and testing image dataset (**30**%)

Normalize the data $X = \frac{z-min(z)}{max(z)-min(z)}$

Train the CNN model:

1: initialize $\boldsymbol{\theta}$

Training Process

2: **for** *numberofepochs* **do**

3: CNN forward pass $\longrightarrow \hat{Y}_i$

4: calculate loss function $L \longrightarrow -\sum_{i=1}^{M} Y_i \log(\hat{Y}_i)$

5: perform back propagation $\longrightarrow \frac{\partial L}{\partial \theta_i}$

6: update CNN weights, $\theta_{i+1} \longrightarrow \theta_i - \eta \frac{\partial L}{\partial \theta_i}$

7: **end for**

8: return $\boldsymbol{\theta}$

Test the CNN model: *Testing Process*

9: **for** *Z in TestingImagedataset* **do**

10: CNN forward pass $\longrightarrow \hat{Y}_i$

11: Accuracy $\longrightarrow \frac{\sum_{i=1}^{N} 1(Y_i == \hat{Y}_i)}{\sum_{i=1}^{N} Y_i}$

12: **end for**

14: **return** $Accuracy$

Security Analysis: The security of the proposed framework, which is judged by how difficult the original image can be recovered from the transmitted compressed image, is analyzed in this section. Assuming the compressed image is intercepted during transmission, it is possible to recover the original image if the pre-trained weights and other parameters of the decoder associated with the intermediate features are known. This method is impossible as the parameters of the decoder are not known as they are not transmitted. The original image can still be recovered by building a model using a dataset of the input image and the corresponding intermediate features as stated in [133, 134]. However, for our proposed framework, this method is impossible as the input image is not available or transmitted. Furthermore, another possible method to recover the original image is by training a decoder using the pre-trained weights of the encoder and other parameters of the autoencoder used in generating the intermediate features. However, the pre-trained weights of the

encoder and other parameters of the autoencoder are not transmitted to the server (only the intermediate feature is transmitted) or known, making this method impossible or very challenging. This method also requires the input original input which is not available. This recovery is a very non-trivial problem as there are infinitely large possible model configurations to train and to check if they can reconstruct the original image. The mathematical proof to show that it is challenging to reconstruct the input image from the compressed image is carried out in [85].

3.2.3.2 Computation offloading

System Model: Without loss of generality, a MEC system with multi-server multiuser is considered, as illustrated in Fig.19. There exist $\mathcal{N}$ mobile users (MUs), i.e., $\mathcal{N} = \{MU_1, MU_2, \dots, MU_N\}$, which can be associated with one MES co-located with an AP at a time. Here, the AP with co-located MES can be considered as equipment has communications and computing capacities, which are so-called computational access points (CAPs). In this architecture, the widely deployed wireless local area network (WLAN) is considered as a potential technology for wireless communications between MUs and the CAPs on the unlicensed frequency band. We define the set of CAPs as $\mathcal{K} = \{1, \dots, K\}$. For example, in Fig. 19, the MU 3 lying in the overlapping coverage areas of CAP 1 and CAP 2 can offload its job to either one of the two CAPs. In this case, each MU not only should determine whether its jobs to be offloaded or processed locally but also which CAP to be o_oaded needs to be considered. In order to be consistent with the practical scenarios, it is assumed that the total computation ability and storage capacity of each CAP is limited and thus cannot be always sufficient for all associated MUs to offload their jobs simultaneously.

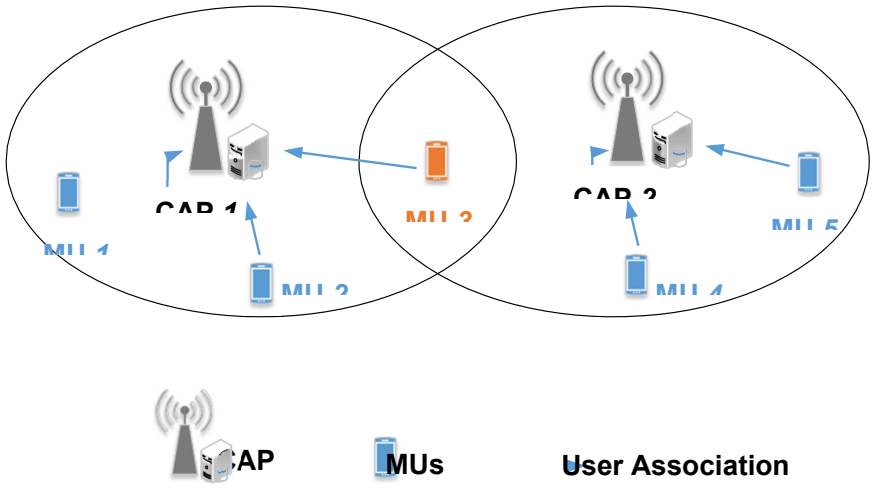


Figure 19: An example of a MEC system with multi-servers, where two CAPs and five MUs are shown

Let $D_n = \{D_{n,1}, \dots, D_{n,K}\}$, $\forall n \in \mathcal{N}$, denote the K-dimensional offloading decision vector of the MU n to CAP k, where $D_{n,k} \in \{0, 1\}$; denotes the computation offloading decision of MU n to the CAP k. Then we have

$$D_{n,k} = \begin{cases} 1 & if\ MU_n \text{ offloads to the CAP } k, \forall k \in \mathcal{K} \\ 0 & \text{otherwise} \end{cases} \tag{3.7}$$

We assume that the MU n either locally processes the job or offloads to one of the associated CAPs, then we have

$$\sum_{k=0}^{K} D_{n,k} = 1, n \in \mathcal{N} \tag{3.8}$$

In addition to the offloading decision, how each CAP allocates its computation resources to the associated MUs should be studied as well. Hence, let $F_n = \{f_{n,1}, \dots, f_{n,K}\}$ denote the K-dimensional allocated computational resource (in central processing unit (CPU) cycles per second) vector to the MU n by the CAP k, the offloading strategy of MU n can be defined as

$$S_n = \{D_n, F_n\} \tag{3.9}$$

Based on definitions above, the offloading decision vector for all the MUs in multiserver MEC system is given as

$$\boldsymbol{S} = \{S_1, \dots, S_n\}, \forall n \in \mathcal{N} \tag{3.10}$$

Job Model: We assume that each MU has only one computation-intensive job (denoted as $\mathcal{J}_n, n \in \mathcal{N}$) to be processed at a time, which is atomic and cannot be further divided. As a result, MU n can only execute it locally or by offloading to the CAP. In order to make the job more visible and intuitive, we characterize the job by a three-tuple of parameters, i.e., $\mathcal{J}_n(s_n, c_n, \vartheta_n)$. In particular, s_n [bits] specifies the amount of input data necessary to be processed, cn [cycles] denotes the amount of computation to accomplish $\mathcal{J}_n$, i.e., the total number of CPU cycles required to process $\mathcal{J}_n$, and ϑ_n [secs] denotes the maximum tolerable delay of $\mathcal{J}_n$. By profiling of the job

execution carefully, the values of s_n, c_n and ϑ_n can be obtained [135]. Obviously, by offloading the computation jobs to the CAPs, the MUs would save their energy for the execution of the jobs. However, additional delay and energy would be introduced for offloading the jobs to the CAPs.

Communication Model: Due to a large amount of computation input data may be uploaded from the MUs to the CAP, abundant wireless spectrum is required, which has become more and more scarce and precious. It is assumed that orthogonal multiple access (OMA) is used as the multiple access scheme in the uplink, e.g., orthogonal frequency division multiple access (OFDMA), [136, 137], which has been adopted in many communication standards. Therefore, for each CAP, the operational frequency band B is divided into N equal sub-bands of size $W_n = B/N$ [Hz], $n \in \mathcal{N}$, and the set of available sub-band at each CAP is denoted as $\boldsymbol{\mathcal{B}} = \{1, \dots, N\}$. Each MU associated to the same CAP is assigned to one non-overlapped sub-band W_n such that intra-cell interference can be avoided in the uplink. However, uplink inter-cell interference may occur among di_erent MUs associated with different CAPs. This could lead to reduced link quality and performance.

In the uplink OFDMA system, the received signal from MU n to the CAP k via the sub-band i is given as

$$y_{(n,k)}^{i} = \underbrace{\sqrt{P_{t(n,k)}^{i}} h_{(n,k)}^{i} x_{(n,k)}}_{Desired\ signal} + \underbrace{\sum_{l \neq k, l \in K} \sqrt{P_{t(n,l)}^{i}} h_{(n,l)}^{i} x_{(n,l)}}_{\text{Intel-cell interferences}} + \underbrace{z_{(n,k)}^{i}}_{Noise}, \forall n \in \mathcal{N}, \forall k \in \mathcal{K}, \forall i \in \boldsymbol{\mathcal{B}}, \tag{3.11}$$

where $x_{(n,k)}$ is the original signal sent from the MU n to CAP k. $h_{(n;k)}^{i}$ denotes the channel power gain for MU n connecting with the CAP k via the sub-band i. It is assumed that the channel remains static within each job offloading procedure, in which the optimal offloading strategy $S_n^* = \{D_n^*, F_n^*\}$ is achieved. $P_{t(n,k)}^{i}$ is the transmit power of the MU n, and the noise power $z_{(n,k)}^{i}$ can be generally considered as the white Gaussian noise in additive white Gaussian noise (AWGN) channel with zero mean and variance δ^2.

Therefore, the received signal-to-interference-plus-noise ratio (SINR) of MU n served by the CAP k on sub-band i is calculated as

$$SINR_{(n,k)}^{i} = \frac{P_{t(n,k)}^{i} \left| h_{(n,k)}^{i} \right|^2}{\delta^2 + \sum_{l=1, l \neq k}^{K} P_{t(n,k)}^{i} \left| h_{(n,k)}^{i} \right|^2} \tag{3.12}$$

where the second term at the denominator indicates the accumulated inter-cell interferences from all the MUs associated with other CAPs on the same sub-band $i;\ \forall i \in \boldsymbol{\mathcal{B}}$.

Computation Model:

For the offloading strategy $S_n^* = \{\boldsymbol{D}_n, \boldsymbol{F}_n\}$ of the MU n, the offloading decision can be "locally" or "offloading", i.e., $D_{n,k} \in \{0, 1\}$. Here, the two computation models are detailed as follows.

Processing locally:

Let τ_l^n be the local execution delay of the job $\mathcal{J}_n$, denote f_l^n as the CPU cycle frequency (i.e., CPU cycles per second) of the MU n. Without loss generality, we assume that the computational capabilities of each device can be different. Then the local execution delay is given as

$$\tau_l^n = \frac{c_n}{f_l^n} \tag{3.13}$$

According to the widely adopted model of the energy consumption [138], the energy consumption processing $\mathcal{J}_n$ with the CPU clock speed f_l^n can be calculated as

$$\varepsilon_l^n = \kappa(f_l^n)^2 c_n, \tag{3.14}$$

where κ denotes the energy efficiency parameter that is mainly depends on the chip architecture [139].

Based on (3.13), (3.14), the weighted-cost for computing $\mathcal{J}_n$ locally is calculated as

$$\mathcal{O}_l^n = \alpha\tau_l^n + (1-\alpha)\varepsilon_l^n, \forall n \in \mathcal{N}. \tag{3.15}$$

where $\alpha, 0 \leq \alpha \leq 1$, specifies the MU's preference on processing delay, and $(1-\alpha)$

specifies the MU's preference on energy consumption. For example, a mobile user with short battery life can decrease the coefficient α so as to save more energy at the expense of longer job processing delay, and vice versa.

Processing via offloading: In case that MU n offloads $\mathcal{J}_n$ to the CAP k, the incurred delay comprises the following three items: (1) the time to upload $\mathcal{J}_n$ to the CAP via the wireless uplink ($T_u^{(n,k)}$

[secs]), (2) the time to execute Jn at the CAP k ($T_p^{(n,k)}$[secs]), which allocates the computational resources accordingly and executes $\mathcal{J}_n$ instead, and (3) the time to transmit the execution result back to the MU n via the wireless downlink ($T_d^{(n,k)}$ [secs]). In the following, we describe the three items in detail.

i) Jobs Uploading. We consider the MEC system with OMA (i.e., OFDMA) as the multiple access scheme in the uplink, in which the MUs can upload their jobs to the CAPs via orthogonal sub-bands simultaneously. Denote $r_u^{(n,k)} = W_n \log_2\left(1 + SINR_{(n,k)}^i\right)$ as the achieved data rate of the wireless uplink from MU n to the CAP k, the delay for uploading job is obtained as

$$T_u^{(n,k)} = \frac{s_n}{r_u^{(n,k)}} = \frac{s_n}{W_n \log_2\left(1 + SINR_{(n,k)}^i\right)}. \tag{3.16}$$

The energy consumption for uploading transmission of MU n to CAP k via the subband i is calculated as $e_u^n = P_{t(n,k)}^i T_u^{(n,k)}/\xi_n$ where ξ_n is the power amplifier efficiency of the MU n, and

$\xi_n = 1$ can hold in general [104]. Thus, the energy consumption during the data uploading can be simplified as

$$e_u^n = P_{t(n,k)}^i T_u^{(n,k)} = \frac{P_{t(n,k)}^i s_n}{W_n \log_2\left(1 + SINR_{(n,k)}^i\right)} \tag{3.17}$$

ii) Jobs Execution. Suppose that the MEC server (MES) located at the CAP can provide computation offloading service to multiple MUs concurrently, so there is no need for queuing in this case [104, 140{142]. In fact, the concurrently processing at the MES can be achieved in practice. For example, the MES may be equipped with a multi-core high-speed CPU [141] or with n-virtual machines (VMs) created by a single CPU [142], so that it can execute n different tasks in parallel and the queuing latency at the MEC server is negligible.

During the execution of the jobs at the CAP, the computing resources made available by the CAP to be shared among the associating MUs are quantified by the allocated computational resources expressed in terms of the number of CPU cycles-per-second, i.e., $f_{n,k}, \forall\, n \in \mathcal{N}, \forall k \in \mathcal{K}$. After receiving the offloaded jobs from the associated MUs, the CAP executes the jobs on behalf of the MUs and then returns the execution result back to them. Therefore, for a feasible computational resource allocation strategy of the CAP k to MU n defined as $\boldsymbol{F}_n = \{f_{n,1}, f_{n,2}, \dots, f_{n,k}\}, \forall k \in \mathcal{K}$. The computing resource constraint should be satisfied, which is expressed as

$$\sum_{n \in \mathcal{N}} f_{n,k} \leq F_k, \forall k \in \mathcal{K}, \tag{3.18}$$

where $\boldsymbol{F}_k$ denotes the entire computational resources of the CAP k, expressed in terms of the number of CPU cycles/sec.

It is worth noting that the allocated computation resources can be either physical computing cores or virtual machine (VM) with moderate computing capabilities provisioned by the MES [104]. Therefore, given a computing resource allocation strategy $\boldsymbol{F}_k$, the execution time to process $\mathcal{J}_n$ by the CAP k is calculated as

$$T_p^{(n,k)} = \frac{c_n}{f_{n,k}}. \tag{3.19}$$

Suppose that MU n stays idle while waiting for the execution results from the CAP, the power consumption of MU n staying the idle state is defined as P_I^n. The energy consumption can be obtained as

$$e_I^n = \frac{P_I^n c_n}{f_{n,k}} \tag{3.20}$$

iii) Results Downloading. Suppose that the symmetric channel is considered, the data rate of the wireless downlink from the CAP k to MU n is calculated as $r_d^{(n,k)} = W_n \log_2\left(1 + SINR_{(k,n)}^i\right)$, where $SINR_{(k,n)}^i = \frac{P_{t(k,n)}^i \left|h_{(n,k)}^i\right|^2}{\delta^2 + \sum_{l=1,l\neq k}^K P_{t(l,n)}^i \left|h_{(n,l)}^i\right|^2}$ denotes the received SINR at the MU n, $P_{t(k,n)}^i$ is the CAP k's transmission power at sub-band i. Denote the size of the execution result of MU n as ω_n, the time to download the executive results from the CAP k is calculated as

$$T_d^{(n,k)} = \frac{\omega_n}{r_d^{(n,k)}}. \tag{3.21}$$

Besides, denote the power consumption of MU n downloading the execution result from CAP k via the sub-band i as $P_{d(n,k)}^i$. The energy consumption of MU n during downloading the results is calculated as

$$e_d^n = \frac{P_{d(n,k)}^i \omega_n}{r_d^{(n,k)}} \tag{3.22}$$

Since in some cases, the size of the execution result is generally much smaller than the size of the job, i.e., $\omega_n \ll s_n$, and the downlink data rate is much higher than that of the uplink, i.e.,

$r_d^{(n,k)} \gg r_u^{(n,k)}$, the delay of downloading the execution result can be omitted [143]. Without loss of generality, in this paper, the total execution delay and energy consumption of MU n are given as

$$\tau_o^n = \frac{s_n}{r_u^{(n,k)}} + \frac{c_n}{f_{n,k}} + \frac{\omega_n}{r_d^{(n,k)}} \tag{3.23}$$

$$\varepsilon_o^n = \frac{P_{t(n,k)}^i s_n}{r_u^{(n,k)}} + \frac{P_l^n c_n}{f_{n,k}} + \frac{P_{d(n,k)}^i \omega_n}{r_d^{(n,k)}} \tag{3.24}$$

According to (3.23) and (3.24), the weighted-cost of MU n for offloading $\mathcal{J}_n$ to the CAP k is given by

$$\mathcal{O}_o^n = \alpha\tau_o^n + (1-\alpha)\varepsilon_o^n, \forall\, n \in \mathcal{N} \tag{3.25}$$

In practice, the coefficient α can be set according to the remaining battery level. For instance, $\alpha = 0$ aims at saving power extremely while $\alpha = 1$ aims at minimizing the offloading delay.

Problem Formulation and Analysis: The expressions of weighted-cost in (3.15) and (3.25) clearly show the interplay between job processing delay and energy consumption aspects, which motivates a joint optimization of offloading decision and computational resources allocation so as to minimize total system cost. In this section, we formulate the optimization problem of joint job offloading decision and computational resource allocation, followed by the problem analysis and decomposition. Given the job offloading strategy $S_n = \{\boldsymbol{D}_n, \boldsymbol{F}_n\}$ for the MU n, we define the sum cost of the MEC system as the weighted-sum cost of all the MUs associated with different CAPs, i.e.,

$$\mathcal{O}_{total} = \sum_{n\in\mathcal{N},k\in\mathcal{K}} \big(1 - D_{n,k}\big)\mathcal{O}_l^n + D_{n,k}\mathcal{O}_o^n \tag{3.26}$$

With $\mathcal{O}_l^n$ in (3.15) and (3.25), respectively, and $D_{n,k} \in \{0,1\}$ specifying the offloading decision of MU n associating with CAP k.

Given the MEC system model, our goal is to develop an optimal job offloading strategy, i.e., $S_n = \{\boldsymbol{D}_n^*, \boldsymbol{F}_n^*\}$, for mobile users that can decide whether to offload the jobs and let CAP allocate appropriate computational resource for each associated MU. Here, we formulate the joint offloading and computational resource allocation as a weighted-sum cost minimization problem (denoted as **P1**), subject to individual delay constraints of the MUs' applications and the computational resource limit of CAP.

P1 (Original problem):

$$\underset{\{\boldsymbol{D}_n, \boldsymbol{F}_n\}}{minimize}\ \mathcal{O}_{total}$$

$$s.t. \quad \boldsymbol{C1}: D_{n,k} \in \{0,1\}, \forall\, n \in \mathcal{N}, \forall\, k \in \mathcal{K} \tag{3.27a}$$

$$\boldsymbol{C2}: \left(1 - D_{n,k}\right)\tau_l^n + D_{n,k}\,\tau_o^n \leq \vartheta_n \tag{3.27b}$$

$$\boldsymbol{C3}: 0 \leq f_{n,k} \leq F_k, \forall n \in \mathcal{N}, \forall k \in \mathcal{K} \tag{3.27c}$$

$$\boldsymbol{C4}: \sum_{n=1}^{N} f_{n,k} \leq F_k, \forall n \in \mathcal{N}, \forall k \in \mathcal{K} \tag{3.27d}$$

This minimization problem **P1** involves finding the optimal job offloading strategy S_n^* for MU n, including the optimal o_oading decision vector $\boldsymbol{D}_n^*$ and the optimal computational resource allocation vector $\boldsymbol{F}_n^*$. The constraints in the formulation of **P1** above are detailed as follows: **C1** shows that MU n can only choose to execute the job $\mathcal{J}_n$ locally or offloading to the CAP k. **C2** makes sure that the time cost to process $\mathcal{J}_n$ should not exceed the maximum tolerable delay ϑ_n. **C3** and **C4** guarantee that the computational resource allocated to MU n and the sum of the computational resources allocated to all the offloading MUs should not exceed the resource limit of the CAP k.

Problem Analysis: The optimization problem **P1** in (3.27) is a mixed-integer nonlinear programming (MINLP) problem and achieving the optimal or sub-optimal solutions usually requires exponential time complexity [144]. Given the input parameters that scales linearly with the number of mobile users (N), we aim to design a MEC system achieving competitive offloading performance with low-complexity. Intuitively, the optimization problem **P1** can be solved by going

through all the combinations of the offloading decision vector ($\boldsymbol{D}_n$) and the computational resource allocation ($\boldsymbol{F}_n$). Denote the optimal offloading decision vector and computational resource allocation vector of MU n as $\boldsymbol{D}_n^*$ and $\boldsymbol{F}_n^*$, then we have

$$S_n^* = \underset{\{\boldsymbol{D}_n, \boldsymbol{F}_n\}}{argmin}\, \boldsymbol{\mathcal{O}}_{total} \tag{3.28}$$

However, because $\boldsymbol{D}_n$ is the binary vector, the resolving of **P1** is usually difficult to tackle [116, 144]. In general, the spatial branch and bound (sBB) method has been considered as a candidate way to solve the MINLP problem [145]. The brief idea of the sBB method is similar to exhaustive search. It uses all integer variables to establish a completed search tree and employ the depth-first search strategy to find the accurate optimal solution [105]. In the sBB algorithm, a hierarchy of nodes represented by a binary tree is created (a.k.a. the sBB tree) and then a pure continuous NLP sub-problem can be formed by dropping the integrality requirements of the discrete variables [146]. As a result, the original optimization problem **P1** becomes the root of the sBB tree. Although the sBB can solve the MINLP problem faster than the exhaustive searching, the large overhead will be introduced into the MEC system due to the time-varying wireless channel conditions and diversity of the MUs' profile. Moreover, the obtained results using the sBB method are sometimes sub-optimal, which degrades the performance of the MEC system.

Problem Decomposition: Through reviewing the structure of the objective function and constraints (i.e., **C1-C4**) of the original problem **P1** in (3.27), it is observed that by temporarily fixing the binary offloading decision variable (i.e., $\boldsymbol{D}_n$), the original problem **P1** with high complexity can be further decomposed into two subproblems with separated objective and constraints by employing the Tammer decomposition method [147]. First of all, the original problem **P1** in (3.27) can be rewritten as

$\widetilde{\text{P1}}$ (Equivalent problem):

$$\min_{\boldsymbol{D}_n} \left(\min_{\boldsymbol{F}_n} \boldsymbol{\mathcal{O}}_{total} \right) \tag{3.29}$$

$$s.t. \boldsymbol{C1} - \boldsymbol{C4}$$

Remark

Remark 1 *Note that the constraints on the offloading decision* (i.e., $\boldsymbol{C1} - \boldsymbol{C2}$) *and the constraints on the computational resource allocation (i.e.,* $\boldsymbol{C3} - \boldsymbol{C4}$*) are decoupled from each other, solving the equivalent problem* $\widetilde{\boldsymbol{P1}}$ *in (3.29) is equivalent to solving the following job offloading (JO) subproblem (**P1.1**) that minimizes weighted-sum cost and the computational resource allocation (CRA) subproblem (**P1.2**) with the fixed offloading decision.*

P1.1 (JO subproblem):

$$\min_{\boldsymbol{D}_n} \boldsymbol{\mathcal{O}}^*_{total} \tag{3.30}$$

$$s.t. \quad \boldsymbol{C1} - \boldsymbol{C2}$$

in which $\boldsymbol{\mathcal{O}}^*_{total}$ is the optimal value function corresponding to the resource allocation problem, presented as:

P1.2 (CRA subproblem):

$$\boldsymbol{\mathcal{O}}^*_{total} = \min_{\boldsymbol{F}_n} \boldsymbol{\mathcal{O}}_{total} \tag{3.31}$$

$$s.t. \quad \boldsymbol{C3} - \boldsymbol{C4}$$

Note that the decomposition from the original problem **P1** in (3.27) to subproblem **P1.1** in (3.30) and subproblem **P1.2** in (3.31) will not change the optimality [104]. Therefore, once the solutions to both the subproblem **P1.1** and the subproblem **P1.2** are obtained, the final optimal solution to the original problem **P1** can be achieved. Instead of the conventional optimization methods usually requiring a long time to converge [145], in this paper, we build a multi-task learning model to predict both $\mathbf{D}^*_{\mathrm{n}}$ and $\mathbf{F}^*_{\boldsymbol{n}}$ more efficiently while ensuring the inference accuracy.

Multi-task Learning for Joint Optimization of Offloading Decision and Resource Allocation: In our MEC system, the offloading decision and computational resource allocation mainly depend on the interplay between communications delay and computing time, and the inherent heterogeneity in terms of mobile devices' computing capabilities, computation job requirements, and capacity

of computing resources at the CAP. Besides, the input parameters to the optimization problem may vary from time to time due to the varying of networks environment and users' applications. The above aspects lead to the original optimization problem **P1** challenging to solve in real-time using conventional optimization algorithms. To address these challenges, in this section, we first present the proposed MTL based feedforward neural network (MTFNN) model jointly predicting the optimal offloading decision and computational resource allocation in the single-server MEC system. After this, the proposed MTFNN model is extended into the multi-server MEC system by using the proposed MTFNN based multi-server offloading algorithm.

Multi-task Learning: Multi-task learning (MTL) is an inductive transfer mechanism focusing on solving multiple learning tasks at the same time. In the MTL, more than one loss function is optimized in general while exploiting commonalities and differences across tasks. Compared to single-task learning methods, only a single architecture is trained in the MTL towards learning several different tasks simultaneously. Compared to training the models separately, the MTL can leverage the domain-specific information that other related tasks. This can result in improved learning efficiency and prediction accuracy for the task-specific models [148, 149].

In general, MTL is typically performed with either hard parameter sharing or soft parameter sharing of hidden layers [150]. The proposed MTFNN model belongs to the hard parameter sharing, which is generally applied by sharing the hidden layers between different tasks but can keep task-specific output layers. With the aid of MTL, the proposed MTFNN model can learn multiple tasks in parallel based on a shared representation which significantly reduces the training complexity and speeds up convergence.

Basic Idea: The proposed MTFNN model based offloading framework is illustrated in Fig. 16. The main idea is to establish a deep feedforward neural network (FNN) to predict the optimal solutions of problem **P1** by offline training a large set of optimal solution samples collected by exhaustively searching[1]. In the FNN, each neuron has incoming connections only from the previous layer and outgoing connections only to the next layer [152]. In particular, the convergence of the training iteration procedure for FNN by use of the gradient method is detailed in [153].

[1] It should be noted that several methods have been proposed to scale up deep neural network training across graphics processing unit (GPU) clusters [151], which helps to reduce the runtime of the offline training.

The proposed offloading framework consists of three aspects: data collection, offline training, and inference, as illustrated in Fig. 16. Note that data collection and training are performed offline only once. After the MTFNN model is trained offline, it can achieve inference online once given a set of input parameters. Specifically, in the data collection, we apply the off-the-shelf exhaustively searching algorithm to achieve the optimal solution of problem **P1** and then collect a sufficient number of data samples.

Remark 2 *Considering that K = 1 holds in the single-server multi-user MEC system, then the offloading decision vector and computation resource allocation vector of all the MUs can be simplified as* $\boldsymbol{D} = \{D_1, \dots, D_n\}$ *and* $\boldsymbol{F} = \{f_1, \dots, f_n\}$*, respectively. Therefore, the offloading strategy vector for all the MUs in single-server MEC system can be given as* $\boldsymbol{S} = \{D, F\}$.

During the offline training, the MTFNN model is established and offline trained using the collected data samples $\{\boldsymbol{D}_g, \boldsymbol{\Theta}_g\}$, where $\mathbf{D}_g$ is the ground truth offloading decision vector for all MUs, $\boldsymbol{\Theta}_g = \boldsymbol{F}_g/F$ denotes the ground truth computational resources allocation ratio vector. After the offline training, the pre-trained MTFNN can be used to predict the solutions of problem **P1** online. The whole procedure of the MTFNN model based offloading scheme that solves the original problem **P1** in a single-server MEC system is summarized in *Algorithm 3*.

In the following, we first introduce the problem mapping of MINLP problem to the multi-task learning domain. Then, we explain the data collection in detail. Finally, we present the offline training and inference procedures.

Problem Mapping: In the MEC system, the two output vectors (i.e., $\mathbf{D}^*$ and $\mathbf{F}^*$) of the original problem P1 are related to each other. Apparently, the predictions of $\mathbf{D}^*$ and $\mathbf{F}^*$ can be considered as two individual machine learning tasks. However, these two learning tasks share the same input parameters, and it is known that learning the two related tasks jointly can get better generalization effect than the learning them individually [154]. In this case, we formulate the problem **P1** as a multi-task learning problem, as shown in Fig. 16. Suppose that there exist L learning tasks $\{\boldsymbol{\mathcal{T}_i}\}_{\boldsymbol{i}}^{\boldsymbol{L}} = 1$ that are related to each other, where $L = 2$ in our proposed MTFNN model. Each learning task $\boldsymbol{\mathcal{T}_i}$ is usually accompanied by a training dataset Di which consists of m_i training samples, i.e.,

$$\boldsymbol{\mathcal{D}_i} = \left\{\boldsymbol{X}_j^{(i)}, \boldsymbol{Y}_j^{(i)}, j = 1, \dots, m_i\right\} \tag{3.32}$$

where $\boldsymbol{X}_j^{(i)}$ is the j-th training instance in $\boldsymbol{\mathcal{T}_i}$, $\boldsymbol{Y}_j^{(i)}$ represents its label.

Algorithm 3 MTFNN based Single-server Offloading

Input: Total number of MUs (N) and the set of MUs' profile (i.e., input parameters set);
Output: Optimal offloading strategy $\boldsymbol{S}^*$**;**
1: Generate training dataset based on *Algorithm 2*.
2: Formulate the optimization problem **P1** as (3.27).
3: Perform offline training using the collected training dataset according to *Algorithm 3*.
4: Input the MUs' parameters to the pre-trained MTFNN model;
5: Predict the optimized offloading decision vector ($\mathbf{D}^*$)and resource allocation vector ($\boldsymbol{\Theta}^*$)
6: Obtain the optimal offloading strategy as $\boldsymbol{S}^* = \{\mathbf{D}^*, \boldsymbol{\Theta}^*\}$

Data Collection: As summarized in *Algorithm 4*, we generate and collect training dataset in the MATLAB environment using a computer with NIVIDA GPU TITAN X (Pascal). We independently generate $4 \times 10^4, 5 \times 10^4, 8 \times 10^4, 10^5, 2 \times 10^5, 2.5 \times 10^5$ and 3×10^5 data samples for $N \in [2, 8]$ in the dataset by traversing all the possible combinations of $\mathbf{D}$ and $\boldsymbol{\Theta}$ with the exhaustive searching algorithm[2], so $\mathbf{D}_g$ and $\boldsymbol{\Theta}_g$ can be obtained for a given set of parameters. During each execution, the network parameters are randomly chosen from the ranges given in Table 4, and the statical parameters are given as follows. The channel bandwidth (W) is 1 MHz, and the white noise power is (δ^2) is 7.9×10^{-13}. The energy efficiency parameter (κ) is set as 1×10^{-28}.

Algorithm 4 Dataset Collection

[2] Considering that the $\boldsymbol{\Theta}$ is a decimal vector which ranges from [0.0; 1.0], in this paper, the interval between traversal values is set to be 0.1. Moreover, in order to speed up the exhaustive searching, we implement the Matlab codes on the NIVIDA GPU.

Initialization: $i = 0$ and dataset is empty, i.e., $\boldsymbol{\mathcal{G}} = \emptyset$;
Iteration:
1: **while** $i <$ dataset size **do**
2: $i \leftarrow i + 1$;
3: Generate input parameters set $(\boldsymbol{X}_i)$ for all devices;
4: Formulate the optimization problem P1 as (3.27);
5: Solve **P1** with exhaustive searching method and record the optimal solution as $\boldsymbol{Y}_i = (\mathbf{D}^*, \mathbf{F}^*)$;
6: Add the i-th input/output pair $\{\boldsymbol{X}_i, \boldsymbol{Y}_i\}$ to $\boldsymbol{\mathcal{G}}$.
7: **end while**

Table 4: Critical Parameters and Definitions

Parameters	**Value range**
The number of devices (N) Data payload size (s)	[2 − 8] [1 − 500] kbits
CPU cycle required to process the data (c)	[3 − 1500] Megacycles
CPU frequency of the device (f_l)	[1Hz − 1GHz]
Weights of delay and energy cost (α, β)	[0.0 −1.0]

The CPU computation capacity of the MES (F) is 2.5 GHz. The transmission power (P_t) and idle power (P_I) of each device are set to be 0:3Wand 0.1 W, respectively [155]. In order to enable the collected data to be applied to our MTFNN model, we preprocess and reshape the dataset as a specific ground truth matrix. The collected dataset is split into 80% for the training phase and the rest 20% for the testing phase.

The data generation by using the exhaustive searching algorithm presented in this paper only aims to provide an alternative way to collect the dataset for offline training and testing. In fact, how to generate dataset does not limit the applicability of the proposed MTFNN based offloading framework. Furthermore, in practice, some heuristic searching algorithms such as genetic algorithm (GA) can help to accelerate data generation [156] and this may not be a big burden for the service provider because they usually already have a lot of historical labeled dataset. In this case, there may not necessary for dataset be generated from scratch.

Offline Training and Online Inference:

The multi-task learning framework including offline training and online inference is highlighted in Fig. 20. The training process is illustrated in *Algorithm 5*. During the offline training, back-

propagation is performed to train the MTFNN model using the collected dataset. Specifically, for the classification problem, the probability of each class is predicted using the Softmax function, i.e., the predicted probability for the i-th class is given as

$$\sigma_m(z) = \frac{e^{z_m}}{\sum_{j=1}^{M} e^{z_j}}, m = 1, \dots, M \tag{3.33}$$

where M is the total number of classes, z is the output of the last fully connected layer.

Algorithm 5 Dataset Collection

1: Build training dataset with *Algorithm 2* ;
2: Train the classifier with loss function $\boldsymbol{l}_c$ given in (3.34);
3: Train the regressor with loss function $\boldsymbol{l}r$ given in (3.36);
4: Achieve the weighted-sum loss function $\boldsymbol{l}$ based (3.37);
5: Tune the weights of each layer using backpropagation until $\boldsymbol{l}$ is minimized

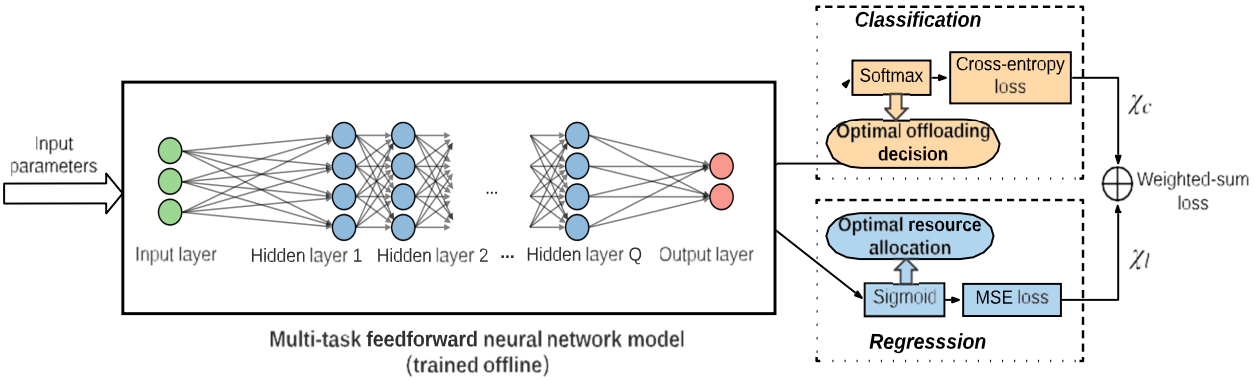


Figure 20: The multi-task learning framework of the proposed MTFNN model

We conventionally set the loss function of the classification (denoted as $\boldsymbol{l}_c$) as crossentropy [157], which is given as

$$\boldsymbol{l}_c = -\frac{1}{M}\sum_{m=1}^{M} \boldsymbol{Y}_m \ln f(\boldsymbol{X}_m), \tag{3.34}$$

where $\boldsymbol{X}_m$ is an input devices' context, $\boldsymbol{Y}_m$denotes the ground truth and $f(\boldsymbol{X}_m)$ is the actual output of neurons.

For the regression problem, the numerical ratio value is mapped between 0 and 1 using the Sigmoid function, i.e., the predicted value is calculated as

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{3.35}$$

Let ω denote the number of input samples, the loss function (denoted as $\boldsymbol{l}_r$) is calculated using mean square error (MSE) [158], i.e.,

$$\boldsymbol{l}_r = \frac{1}{\omega}\sum_{i=1}^{\omega}(\boldsymbol{Y}_i - f(\boldsymbol{X}_i))^2 \tag{3.36}$$

In our proposed MTFNN model, the loss function is defined as the weighted-sum of $\boldsymbol{l}_c$ and $\boldsymbol{l}_r$, i.e.,

$$\boldsymbol{l} = \chi_c \boldsymbol{l}_c + \chi_r \boldsymbol{l}_r \tag{3.37}$$

where χ_c and χ_r denote the weights. Here, the Adam optimizer [159] is used to optimize the MTFNN model via performing back-propagation.

Remark 3 *In the proposed MTFNN model, joint optimization is performed to minimize the loss function including the MSE loss and the cross-entropy loss. There are two weights (χ_c and χ_r) for the two losses and the importance of the tasks in multi-task learning can be determined by the weights.*

With the pre-trained MTFNN model, the optimal offloading strategy ($\boldsymbol{D}^*$ and $\boldsymbol{\Theta}^*$) can be predicted using online inference. Taking the pre-trained MTFNN model with $\chi_c = \chi_r = 1$ and N = 3 as an example, the input layer contains 18 neurons, the three hidden layers contain 60, 40 and 20 neurons, respectively. In the output layer, the classification output contains 8 neurons and the regression output contains 3 neurons. The weights are initialized using Xavier initialization and the hidden layers use the ReLU activation function. Furthermore, the Batch normalization is added after the first and third hidden layers. Adam optimizer is used and turned with parameters, i.e., learning rate is 0:001, beta1 = 0:9, beta2 = 0:999. The deep learning model is trained for 50 epochs with batch size of 64. During the inference, given each MU's parameters, the pre-trained MTFNN model can predict the offloading decision vector $\boldsymbol{D}^*$ and the resource allocation vector $\boldsymbol{\Theta}^*$ accordingly by performing the feed-forward calculation.

MTFNN Model Applied in Multi-server MEC System:

In a multi-server MEC system, we consider the scenario with N MUs, which can be associated with K CAPs, as shown in Fig. 3.17. The set of MUs and CAPs is denoted as $\mathcal{N} = \{MU_1, MU_2, \dots, MU_N\}$ and $\mathcal{K} = \{1, \dots, K\}$, respectively. All CAPs are assumed to be connected to a wired backhaul controller (denoted as C), which can help to coordinate resource allocation among different CAPs to MUs in near-real-time [160, 161].

Different from the single-server MEC system where only the optimal offloading strategy for each MU is optimized, in the multi-server MEC system, the MUs located in the overlapping coverage areas of different CAPs further need to choose an optimal CAP to associate with to minimize the total system cost. Therefore, compared to the single-server MEC system, the optimization problem in multi-server MEC system is to find an optimal offloading strategy including an optimal CAP association. To solve this optimization problem, an MTFNN based multi-server offloading algorithm is proposed, as illustrated in *Algorithm 6*.

In *Algorithm 6*, each CAP (e.g., CAP k; 8k 2 K) executes the MTFNN based single-server offloading algorithm (i.e., Algorithm 3) separately under each overlapped MUs' association case. Denote the total number of the overlapped MUs' association cases as $L, L \leq K^U$, where K is the number of CAPs and U denotes the total number of MUs located at the overlapping area of CAPs. In other words, *Algorithm 3* is performed by each CAP at most L times. After that, the controller selects the optimal offloading strategy achieving the minimal total cost. To further reduce the number of times the algorithm running unnecessarily, each MU located at the overlapping area of CAPs can first evaluate the signal strength from the potential CAPs and then generates the candidate associated CAPs set including total $W, W \leq K$, CAPs with relatively high signal strength. In this case, all the combinations of U MUs with W candidate associated CAPs can be obtained as $L \leq W^U$.

Algorithm 6 MTFNN based Multi-server Offloading

Input: Total number of CAPs (K), total number of MUs (N), number of MUs located at the overlapping area of CAPs (U) and the set of MUs' profile (i.e., input parameters set);

Output: Optimal offloading strategy $\boldsymbol{S}^*$;

1: Each MU located at the overlapping area of CAPs evaluates the signal strength from the potential CAPs and then generates the candidate associated CAPs set including total W

CAPs with high signal strength, where $W \leq K$

2: Generate all the combinations from each MU to W candidate associated CAPs. Denote the total number of combinations as $L, L \leq W^U$;

3: **while** $j < L$ **do**;

4: $j \leftarrow j + 1$;

5: **while** $k < K$ **do**;

6: Input the MUs' parameters to the pre-trained MTFNN model deployed at the CAP k;

7: The CAP k predicts MU n's offloading decision vector $\left(\boldsymbol{D}^k = \left\{D_{1,k}, \ldots, D_{N,k}\right\}\right)$ and resource allocation vector $\left(\boldsymbol{\Theta}^k = \left\{\boldsymbol{\Theta}_{1,k}, \ldots, \boldsymbol{\Theta}_{N,k}\right\}\right), \forall k \in \mathcal{K}$;

8: The CAP k obtains the offloading strategy, i.e., $S^{j,k} = \{\boldsymbol{D}^k, \boldsymbol{\Theta}^k\}$;

9: $k \leftarrow k + 1$;

10: **end while**

11: Each CAP uploads the predicted offloading strategy $S^{j,k}$ to the controller $\boldsymbol{\mathcal{C}}$;

12: $\boldsymbol{\mathcal{C}}$ combines the offloading strategy of each CAP and then obtains the offloading strategy of MU n, i.e., $S_n^j = \left\{\boldsymbol{D}_n^j, \boldsymbol{\Theta}_n^j\right\}$;

13: $\boldsymbol{\mathcal{C}}$ calculates the total system cost based on (3.26), denoted as $\mathcal{O}_{total}^j$;

14: **end while**

15: $\boldsymbol{\mathcal{C}}$ selects the optimal offloading strategy from all the association combinations as $\boldsymbol{S}^* = \{S_n^*\}$, where $S_n^* = \{\boldsymbol{D}_n^*, \mathbf{F}_n^*\}$, and the total cost using the offloading strategy $\boldsymbol{S}^*$ meets the condition $\mathcal{O}_{total}^* = \min\left\{\mathcal{O}_{total}^j\right\}, \forall j \in [1, L]$;

16: $\boldsymbol{\mathcal{C}}$ broadcasts the optimal offloading strategy $\boldsymbol{S}^*$ to all the CAPs, and then the CAPs inform the associated MUs the optimal offloading strategy.

3.3 Knowledge Extraction using Machine Learning and Deep Learning

3.3.1 State-of-The-Art

3.3.1.1 Evidence Theory for Big Data Processing

Dempster-Shafer theory of evidence (DST) [162] has been used to combine data (called evidence) from multiple sources. Compared to traditional Bayesian method, Dempster-Shafer theory has more flexibility in specifying ignorance and uncertainty in the data. When conflicts level among source of data become large and the refinement of frame of discernment is inaccessible because of the vague and imprecise nature of elements of frame of discernment [163], Dezert-Smarandache theory (DSmT) [164] can be applied as a powerful tool to combine the data. However, the methods in both DST and DSmT frameworks are in general very computationally expensive when the number of hypotheses and evidences increase, thus in many data fusion applications with big data processing, they may not be directly applied to multiple data sources with high cardinality.

3.3.1.2 Semi-supervised Learning

Supervised learning requires large amount of labeled data. However, the labeled data is usually not available or at least not in large amount in practice because labeling data is labor intensive and there may not be enough time to label large amount of data in real time. An example of such use cases is fake news detection. Fake news detection has attracted a lot of attention in recent years. There are extensive studies such as content based method and propagation pattern based method. Content based method typically involves two steps: preprocessing news contents and training supervised learning model on the preprocessed contents. The first step usually involves tokenization, stemming, and/or weighting words [165, 166]. In the second step, Term Frequency-Inverse Document Frequency (TF-IDF) [167,168] may be employed to build samples to train supervised learning models. However, the samples generated by TF-IDF will be sparse, especially for social media data. To overcome this challenge, word embedding methods such as word2vec [169] and GloVe [170] are used to convert words into vectors.

In addition, Mihalcea *et al*. [171] used linguistic inquiry and word count (LIWC) [172] to explore the difference of word usage between deceptive language and nondeceptive ones. Specifically, deep learning based models have been explored more than other supervised learning models [173]. For example, Rashkin *et al*. [174] built the detection model with two LSTM RNN models: one learns on simple word embeddings, and the other enhances the performance by concatenating long short-term memory (LSTM) outputs with LIWC feature vectors. Doc2vec [175] is also applied to represent content that is related to each social engagement. Attention based RNNs are employed to achieve better performance as well. Long et al. [176] incorporates the speaker names and the statement topics into the inputs to the attention based RNN. In addition, convolutional neural networks are also widely used since they succeed in many text classification tasks. Karimi *et al*. [177] proposed Multi-source Multi-class Fake news Detection framework (MMFD), where CNN analyzes local patterns of each text in a claim and LSTM analyze temporal dependencies in the entire text.

In propagation pattern based method, the propagation patterns have been extracted from time-series information of news spreading on social media such as Twitter and they are used as features for detecting fake news. For instance, to identify fake news from microblogs, Ma *et al*. [178] proposed the Dynamic Series-Time Structure (DSTS) to capture variations in social context features

such as microblog contents and users over time for early detection of rumors. Lan *et a*l. [179] proposed Hierarchical Attention RNN (HARNN) that uses a Bi-GRU layer with the attention mechanism to capture high-level representations of rumor contents, and a gated recurrent unit (GRU) layer to extract semantic changes. Hashimoto et al. [180] visualized topic structures in timeseries variation and seeks help from external reliable source to determine the topic truthfulness.

In summary, most of the current methods are based on supervised learning. It requires a large amount of labeled data to implement the detection processes, especially for the deep learning based approaches. However, annotating the news on social media is too expensive and costs a huge amount of human labor due to the huge size of social media data. Furthermore, this is almost impossible to achieve in near real time. Even with labeled data, constructing the huge amount of labeled corpus is an extremely difficult task in the field of natural language processing as it costs a large volume of resources and it is challenging to guarantee the label consistence. Therefore, it is imperative to incorporate unlabeled data together with labeled data in fake news detection to enhance the detection performance. Semi-supervised learning [181, 182] is a technique that is able to use both labeled data and unlabeled data.

Semi-supervised learning (SSL) is to employ unlabeled data to enhance the model performance through learning on a small scale of labeled data and a large scale of unlabeled data [182]. For text classification, it is able to contribute to improving classification performance for emerging topics that are lacking of labeled data. For example, Zhu *et al*. [183] proposed label propagation (LP) through performing Markov random walks [184] on a graph, which has been employed for text classification such as Twitter polarity classification [185] and fake news detection [186]. Chapelle *et al*. [187] utilized low density separation to implement semi-supervised text classification through combining graph distance computation with transductive support vector machine (TSVM). Nigam *et al*. [188] combined expectation maximization (EM) and generative models to build semi-supervised text classifers. Shi *et al*. [189] employed transfer learning for completing semi-supervised learning for text classification, where EM algorithm is employed to measure the correlations between words. Zhao et al. [190] developed semi-supervised frequency estimate (SFE) for large scale text classification. Recently, deep learning models have been used to build

semi-supervised learning models for text classification such as sentiment analysis [191]. In addition, the clustering technique [192] is applied to building semi-supervised machine learning models, where the clustering is to determine if there are more than one class labeled on one cluster, and to examine if there is no labeled data point in one cluster [193].

3.3.2 Motivation

3.3.2.1 Evidence Theory for Big Data Processing

The Internet of Things (IoT) has brought the vision of a more connected world into reality with big data analytics and numerous services, which can help individuals, businesses, and society on a daily basis. IoT opens a new horizon of ubiquitous sensing, interconnection of devices, service sharing, and provisioning to support better communication and collaboration among people and devices in a more distributed and dynamic world. This new paradigm also supports advanced processing of large IoT data streams, as well as to provide automated decision making in near real time.

In many IoT applications, multiple sensors are deployed to monitor a phenomenon that can be modeled by multi hypotheses. The goal is to detect and determine the current related hypothesis among possible multi hypotheses [194].

In Radio Frequency IDentification (RFID) systems to enhance the system performance a distributed overlap aware technique used to eliminate redundant RFID reader [195]. In Wireless Sensor Networks (WSN), to traffic anomaly detection (well known black hole attack) [196] proposed a profile based monitoring approach with a restricted feature set. In NoSQL database systems, [197] designed a data analytics tool that enables knowledge discovery through information retrieval from document-append style storage.

In reality sensors are far from perfect. In a sensor field, sensors provided by different manufacturers may have different specifications, accuracy and sensitivity range. Moreover, their functionalities decay along the years. Based on sensors' location, sensitivity range, and their distance from the source of the event, they will be affected differently. Furthermore, most type of sensors are not omni-directional and they are only sensitive when their sensing window directed to the source of the event. According to sensors specification sheets, most type of sensors are sensitive to more than one parameter. For example, carbon monoxide (CO) gas sensors can response to other gases

as well. Thus it is not certain about the values measured by those sensors. Environmental interference and noise can affect their accuracy and measurements as well.

In future smart buildings or smart environment, numerous sensors will be deployed for monitoring and surveillance. As a result, large amount of data will be collected from various sources. In many practical cases, the data may contain uncertainties and sometimes even are conflicting. Since the readings from multiple sensors are from different sources, plus the uncertainties and conflicting data, how to use the big uncertain data to make inference and decisions becomes a challenge.

In order to increase the amount of global information while decreasing its uncertainty, novel data processing methods are indispensable to improve the quality of decision making by taking a vantage of information redundancy and complementariness among sources. In this work, Dempster-Shafer Theory (DST) [162, 198], [199] based methods are proposed to combine the data (evidences) from different sensors [200]. In the case of conflict among the sources, Dezert-Smarandache Theory (DSmT) can be a better solution [201].

In this work, we explore the feasibility of using DST and DSmT theories in practical applications with high number of multi hypothesis through a case study. Specifically, we propose a modified algorithm to use DST and DSmT with reduced computational cost to analyze temperature and humidity data received from multiple sensors to determine comfort zones in a smart building. Comfort zone is defined as the range of temperature and humidity that people are feeling comfortable. It is known as a thermal/human comfort too. Evaluating comfort zone is related to different parameters and even different from person to person. Figure 21 shows the “Comfort Zone” according to ISO7730-1984 standard. It is designed based on several experiments and a large amount of empirical data that collected over several years from different locations. As these graphs display, comfort zone is different for winter and summer seasons.

In traditional buildings, the sensors are installed in some fixed places and they may not be able to measure at locations of interest. The authors of [202] proposed a novel framework of an environment air quality monitoring system based on community sensing, see Figure 22. Leveraging on the high penetration of smartphones and low cost and small form factor of certain sensors with a Bluetooth module, critical measurements such as air quality can be measured by each sensor carried by a member of a community, and be sent to that person's smartphones, and eventually uploaded to server or cloud using a corresponding app. Then the aggregated data at the server side

can be processed to determine comfort zone and control HVAC (Heating, ventilation, and air conditioning) system to optimize the usage of electricity, while keeping the inhabitants comfortable. In this project, we have designed the architecture of the proposed community sensing system, and implemented the system using commercial off-the-shelf (COTS) Sensordrone, paired with Android© smartphones. Our system measures temperature, humidity, pressure, carbon monoxide, and battery charge level in real-time and it provided the experimental data in this study.

In this paper, we start by introducing the details of Dempster-Shafer theory of evidence (DST) and Dezert-Smarandache theory (DSmT) of plausible and paradoxical reasoning. Then we propose our models for structured high order multi hypotheses and apply different combination rules to calculate total mass, belief, plausibility and pignistic probability. The decision making based on those metrics are used to compare for different models and combination rules. We also analyze the computational complexity between DST and DSmT.

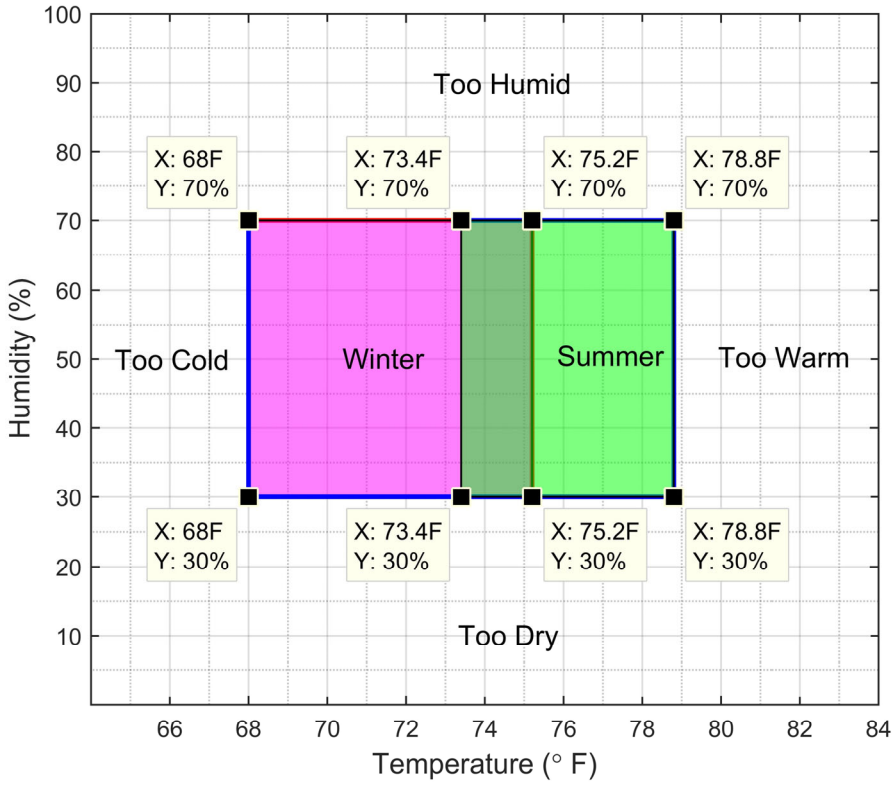


Figure 21: Relative humidity/temperature comfort zone (ISO7730-1984)

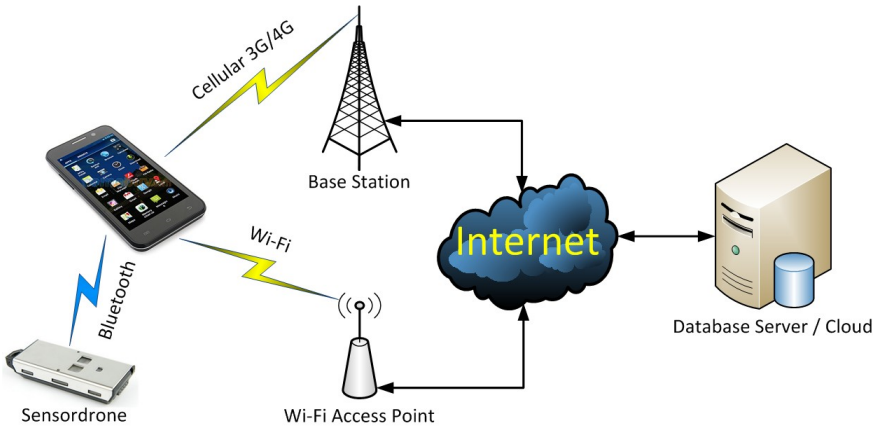


Figure 22: The architecture of the community based sensing system

3.3.2.2 Semi-supervised Learning

Social media (e.g., Twitter and Facebook) has become a new ecosystem for spreading news [203]. Nowadays, people are relying more on social media services rather than traditional media because of its advantages such as social awareness, global connectivity, and real-time sharing of digital information. Unfortunately, social media is full of fake news. Fake news consists of information that is intentionally and verifiably false to mislead readers, which is motivated by chasing personal or organizational profits [204]. For example, fake news has been propagated on Twitter like infectious virus during the 2016 election cycle in the United States [205, 206]. Understanding what can be done to discourage fake news is of great importance.

One of the fundamental steps to discourage fake news would be timely fake news detection. Fake news detection [207{209] is to determine the truthfulness of the news by analyzing the news contents and related information such as propagation patterns.

It attracts a lot of attention to resolve this problem from different aspects, where supervised learning based fake news detection dominates this domain. For instance, Ma *et.al* detects fake news with data representations of the contents that are learned on the labeled news [210]. Early detection is also an effective approach to recognize fake news by identifying the signature of text phrases in social media posts [211]. Moreover, temporal features play a crucial role in the fast-paced social media environment because information spreads more rapidly than traditional media [212]. For example, detecting a burst of topics on social media can capture the variations in temporal patterns

of news [213, 214]. Specifically, deep learning based fake news detection achieves the state-of-the-art performance on different datasets [173], where both recurrent neural networks (RNN) and convolutional neural networks (CNN) are employed to recognize fake news [176, 208, 215]. However, since news spreads on social media at very high speed when an event happens, only very limited labeled data is available in practice for fake news detection, which is inadequate for the supervised model to perform well.

As an emerging task in the field of natural language processing (NLP), fake news detection requires big labeled data to meet the requirement of building supervised learning based detection models. However, annotating the news on social media is too expensive and costs a huge amount of human labor due to the huge size of social media data. Furthermore, this is almost impossible to achieve in near real time. In addition, it is difficult to ensure the annotation consistency for big data labeling [216]. With the increment of the data size, the annotation inconsistence will be worse. Therefore, using unlabeled data to enhance fake news detection becomes a promising solution and more urgent.

In this paper, we propose a deep semi-supervised learning framework with Word CNN [217], by building two-path convolutional neural networks to accomplish timely fake news detection in the case of limited labeled data, where the framework is shown in Figure 23.

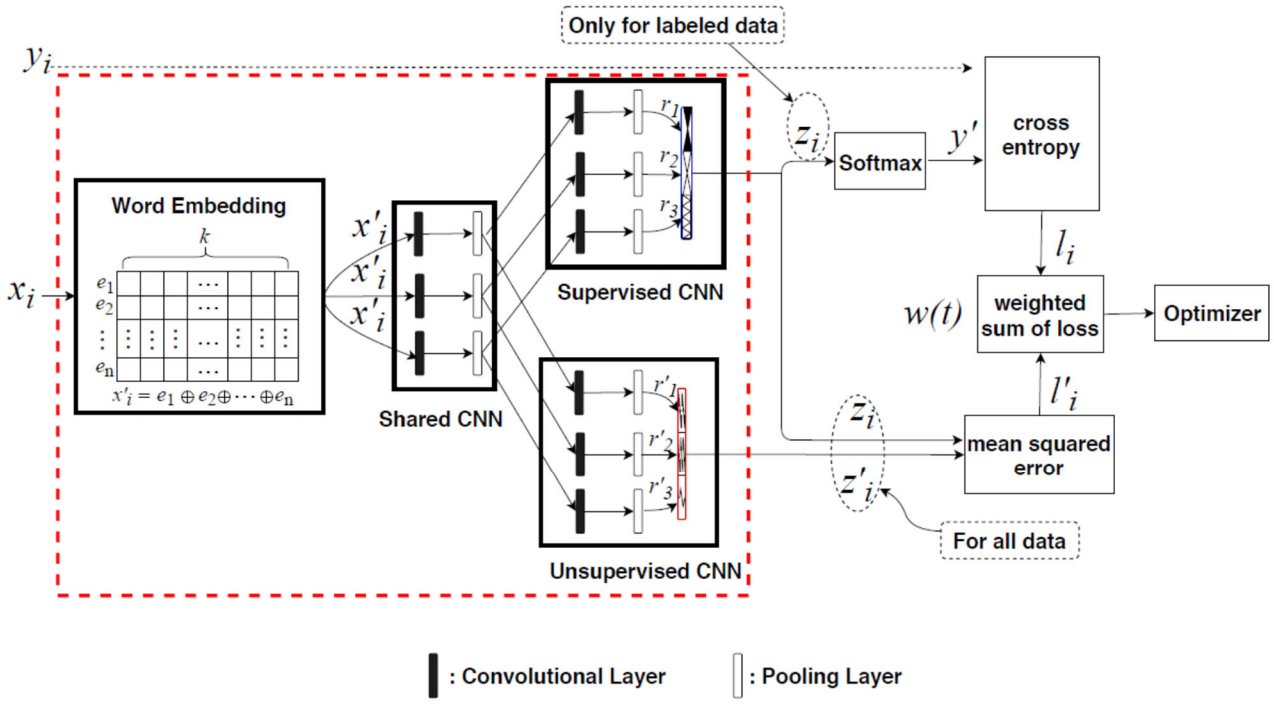


Figure 23: Word-CNN Based Deep Semi-supervised Learning. In the shared CNN, each convolutional layer contains 100 (3×3) filters, 100 (4×4) filters, and 100 (5×5) filters, respectively. Both the supervised CNN and the unsupervised CNN have the same architecture of the shared CNN with different numbers of filters, where each convolutional layer contains 100 (3×3) filters. We use (2×2) max-pooling for all pooling layers. $\oplus$ is the concatenation operator. r_1, r_2 and r_3 are outputs from the supervised path while r_1', r_2' and r_3' are those from the unsupervised path. Furthermore, we concatenate r_1, r_2 and r_3 to conduct z_i and connect to generate z_i'.

It consists of three components, namely, a shared CNN, a supervised CNN, and an unsupervised CNN. One path is composed of the shared CNN and supervised CNN while the other is made of the shared CNN and unsupervised CNN. Moreover, the architectures of these three CNNs can be similar or different, which are determined by the application and performance. All data (labeled and unlabeled data) will be used to generate the mean squared error loss, while only labeled data will be used to calculate the cross-entropy loss. Then a weighted sum of these two losses is used to optimize the proposed framework. We validate the proposed framework on detecting fake news using two datasets, namely, LIAR [208] and PHEME [218]. Experimental results demonstrate the effectiveness of the proposed framework even with very limited labeled data.

3.3.3 Problem Formulation and Proposed Approach

3.3.3.1 Evidence Theory for Big Data Processing

Dempster-Shafer theory

Dempster-Shafer theory (DST) of evidence, or DST, is firstly originated by Dempster's work [219] on the upper and lower probabilities and later extended by Shafer's work [162] on the belief functions. It is an extension of the traditional Bayesian probability that gives capability to deal with uncertainty. To better understand Dempster-Shafer theory, we firstly introduce some propositions [199]:

Frame of discernment: let Θ be a finite set of elements. Elements here refer to hypotheses or classes that for our case study related to feeling zones. Θ called the frame of discernment (FOD). For Dempster-Shafer model, all elements of Θ are assumed be exclusive and exhaustive. The power set of Θ that includes all subset of Θ is defined by 2^{Θ}. Basically power set includes all the elements of Θ and all combinations of their union. So it is closed under union operator.

Mass Function: mass function or basic belief assignment (bba) m is defined as a probability function. It maps a number in [0,1] to elements of 2^{Θ} in such a way that:

$$m: 2^{\Theta} \longrightarrow [0, 1] \tag{3.38}$$

$$m(\emptyset) = 0 \tag{3.39}$$

$$\sum_{A \subseteq 2^{\Theta}} m(A) = 1 \tag{3.40}$$

Here $m(A)$ refers to the level of confidence in A, where A is a subset of 2^{Θ}. In our study, mass function refers to degree of belief for each class of feeling. In the case $m(A) > 0$, subset A is called a focal element. For the case subset A includes more than one element, because we do not have more information about each element separately, related mass function $m(A)$ cannot be decomposed to more mass functions for each individual element. One of the main differences between

traditional Bayesian probability and Dempster-Shafer theory is the uncertainty function $m(\Theta)$ in DST:

$$m(\Theta) = 1 - \sum_{A \subset 2^{\Theta}} m(A)$$

Combination rule of Dempster-Shafer: In many multi sources and big data applications, different types of data are aggregated from multiple sensors that may originated from multiple sources. Combined mass function can be calculated based on the Dempster's rule of combination:

$$m(A) = m_1 \oplus m_2 \oplus \ldots \oplus m_N \tag{3.42}$$

$$m(A) = \begin{cases} 0, & A = \emptyset \\ \dfrac{\sum_{\cap_{k=1}^{N} A_k = A} \prod_{i=1}^{N} m_i(A_i)}{1 - K}, & A \neq \emptyset \end{cases} \tag{3.43}$$

$$K = \sum_{\cap_{k=1}^{N} A_k = \emptyset} \prod_{i=1}^{N} m_i(A_i) \tag{3.44}$$

$$1 - K = \sum_{\cap_{k=1}^{N} A_k \neq \emptyset} \prod_{i=1}^{N} m_i(A_i) \tag{3.45}$$

Here K is the conflict value among all the sources of information. It is used as a normalization factor, $K \in (0, 1)$. The higher value of K indicates more conflicting among information sources. And weight of conflict define as:

$$W = -\log(1 - k) \tag{3.46}$$

As an example, for two sensors, Dempster's rule of combination is:

$$m(A) = m_1 \oplus m_2 \tag{3.47}$$

$$m(A) = \begin{cases} 0\ , & A = \emptyset \\ \dfrac{\sum_{A_1 \cap A_2 = A} m_1(A_1).m_2(A_2)}{1-K}, & A \neq \emptyset \end{cases} \tag{3.48}$$

$$K = \sum_{A_1 \cap A_2 = \emptyset} m_1(A_1).m_2(A_2) \tag{3.49}$$

$$1 - K = \sum_{A_1 \cap A_2 \neq \emptyset} m_1(A_1).m_2(A_2) \tag{3.50}$$

Dempster's rule of combination is associative, commutative and markovian. For the cases with more than two sources of data (called evidences in DST), DST combination rule can be extended by applying combination rule between two mass functions and then combine the result with new evidences and so on to compute combination for all sources of evidences. Associated with mass function, the belief function is defined as:

$$Bel(x) = \sum_{y \in 2^{\Theta}, y \subseteq x} m(y) \tag{3.51}$$

Where x and y are subsets of power set. And plausibility function calculate as:

$$Pl(x) = \sum_{y \in 2^{\Theta}, x \cap y \neq \emptyset} m(y) = 1 - Bel(\bar{x}) \tag{3.52}$$

where $\bar{x}$ is the complement set of x, $\bar{x} = \Theta - x$. It is clear that $Pl(A) \geq Bel(A)$. Belief interval, $[Bel(A);\ Pl(A)]$, refers to the imprecision on the true probability, when belief function is the lower probability and plausibility function as an upper probability.

The pignistic probability introduced by [220] is defined as:

$$bet\ P\ (x) = \sum_{y \in 2^{\Theta},\ y \neq \emptyset} \frac{|x \cap y|}{|y|} . m(y) \tag{3.53}$$

where $|x|$ is the cardinality of x. Pignistic probability maps belief to probability to make a hard decision. As a result, belief functions provide a pessimistic view while plausibility function is optimistic. Pignistic probability is a compromise.

Reliable decision making using big data fusion is a challenge. Although there is not any unique metric for best decision making, four different metrics including total mass function, belief, plausibility and pignistic probability are tested in our simulation and experiment.

Dezert-Smarandache Theory

Dezert-Smarandache theory of plausible and paradoxical reasoning (DSmT) is an extension of DST and a generalized version of both DST and traditional Bayesian probability. DSmT has better performance when the uncertainty or conflicts among evidences are high. In DSmT, hyper power set of Θ is defined by D^{Θ}. It includes all the elements of Θ and all combinations of their union and intersection. Thus DSmT is closed under both union and intersection operators, while DST is closed under union operator only. Unlike DST, in DSmT we are not limited for exclusivity among elements of Θ. It is clear that the cardinality of hyper power set is much more than power set. Similar to DST, in DSmT mass function or generalized basic belief assignment (gbba) is defined as a mapping $m: D^{\Theta} \rightarrow [0,1], m(\emptyset) = 0$ and $\sum_{A \subseteq D^{\Theta}} m(A) = 1$. Belief, plausibility and generalized pignistic probability functions are defined as [164]:

$$Bel(x) = \sum_{y \in D^{\Theta}, y \subseteq x} m(y) \tag{3.54}$$

$$Pl(x) = \sum_{y \in D^{\Theta}, x \cap y \neq \emptyset} m(y) = 1 - Bel(\bar{x}) \tag{3.55}$$

$$bet\ P(x) = \sum_{y \in D^{\Theta},\ y \neq \emptyset} \frac{|\mathcal{C}_{\mathcal{M}}(x \cap y)|}{|\mathcal{C}_{\mathcal{M}}(y)|} . m(y) \tag{3.56}$$

Where $|\mathcal{C}_{\mathcal{M}}(y)|$ is the cardinality, i.e., the number of parts x has in the model (Venn diagram). Several combination rules have been developed based on DSmT model [164]. Those rules can manage or redistribute conflict values in different ways and have different complexity of computation. There are numerous combinations rules can be defined to redistribute conflict values among elements of hyper power set. Classic DSm rule of combination, hybrid DSm rule, and series of proportional conflict redistribution rules (PCR) from PCR1 to PCR6 are some of those combination rules [164] PCR5 is one of the most accurate rules in managing conflict. It redistributes partial conflict values just between the two elements that involved in that partial conflict. However, comparing to other methods it is hard to implement due to high computational cost. For two sources of evidences: $\forall X \in D^{\Theta} \backslash \{\emptyset\}$

$$m_{PCR5}(X) = m_{12}(X) + \sum_{Y \in D^{\Theta}, X \cap Y = \emptyset} \frac{m_1(X)^2 . m_2(Y)}{m_1(X) + m_2(Y)} + \frac{m_2(X)^2 . m_1(Y)}{m_2(X) + m_1(Y)} \tag{3.57}$$

where m_{12} refers to conjunctive consensus:

$$m_{12}(X) = \sum_{X_1, X_2 \in D^{\Theta}, X_1 \cap X_2 = X} m_1(X_1) . m_2(X_2) \tag{3.58}$$

PCR5 can be applied to more than two data sources [221]. Figure 24 shows the flowchart of applying DSmT combination rule (PCR5 as an example here) from sensing data to decision making.

Except the classic DSm rule of combination, all other combination rules based on DSmT model are non-associative and non-markovian. This implies that for more than two sources of evidences, combination rule cannot be applied blindly between two mass functions in repetitive way

that we do in DST. Hence the order of sources in combination can change the result of combination. For calculating PCR5 rule for more than two sources we adopt a new method introduced in [222] to conserve the associativity and markovian property requirement to guarantee the correctness of the final combination. In fact, applying this algorithm transfers a non-associative and non-markovian rule to a quasi-associative and quasi-markovian rule.

To implement PCR5 rule of combination based on this algorithm for $n \geq 3$ sources, first the conjunctive rule, $m_{12}(X)$, calculated between first two sources and the whole conflict mass transfered to empty or non-empty set (we used non empty set) and save the result. Then conjunctive rule calculated between the saved results with the third source. We repeat this for first $n-1$ sources. Finally, PCR5 rule applied between the conjunctive result among $n-1$ sources and the last source. This algorithm has the advantage that the order of sources in the combination rule is no longer important and both associative and markovian properties are satisfied as well.

3.3.3.2 Semi-supervised Learning

We propose a general framework of deep semi-supervised learning and apply it to accomplish fake news detection. Suppose the training data consist of total N inputs, out of which M are labeled. The inputs, denoted by x_i, where $i \in 1 \dots N$, are the news contents that contain sentences related to fake news. In general, the news on social media normally contains limited number of words like 100 or less. It will lead to the data sparsity if we apply TF-IDF to extract features. To relieve the data sparsity problem, we employ word embedding techniques, for instance, word2vec [169,223,224], to represent the news contents. Here S is the set of labeled inputs, $|S| = M$. For every $i \in S$, we have a known correct label $y_i \in 1 \dots C$, where C is the number of different classes.

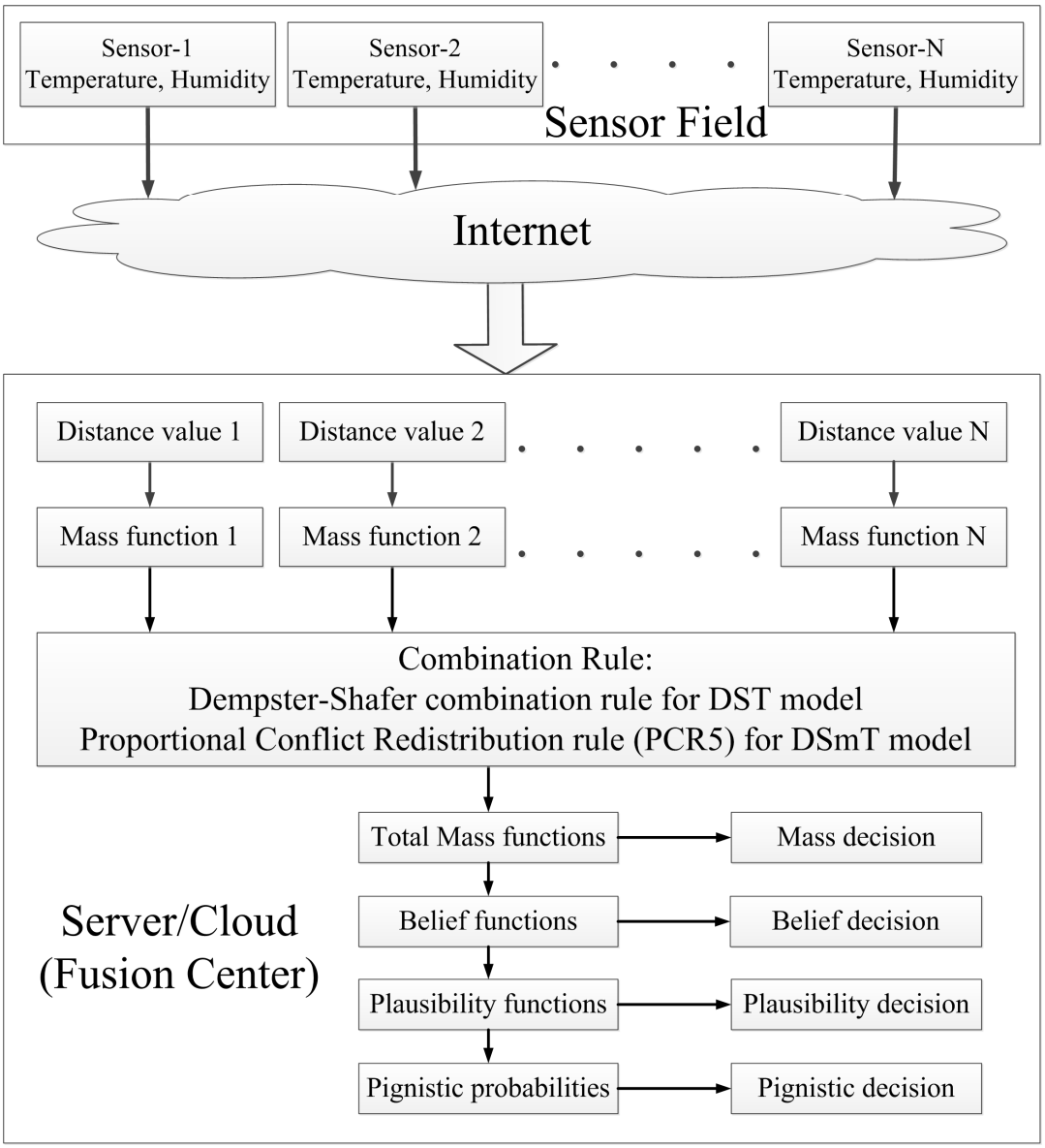


Figure 24: The proposed DST and DSmT decision making platform for multiple data sources

The proposed framework of deep semi-supervised learning and corresponding learning procedures are shown in Figure 23 and Algorithm 7, respectively. The framework is built based on Word CNN [217] that is a powerful classifier with the simple architecture of CNN for sentence classification [217]. Considering fake news detection, let $e_i \in \mathbb{R}^k$ be the k-dimensional word vector corresponding to the i-th word of the sentence in the news. A sentence of length n is represented as $x_i' =$

$e_1 \oplus e_2 \oplus \ldots \oplus e_n$, where $\oplus$ is the concatenation operator. A convolution operation involves a filter $c \in \mathbb{R}^{hk}$, which is applied to a window of h words to produce a new feature. The pooling operation deals with variable sentence lengths.

Algorithm 7 Learning in the proposed framework

1: $x_i =$ training sample;
2: $S =$ set of training samples;
3: $y_i =$ label for labeled x_i $i \in S$;
4: $f_{embedding}(x) =$ word embedding;
5: $f_{\theta_{shared}}(x) =$ shared CNN with trainable parameters θ_{shared};
6: $f_{\theta_{sup}}(x) =$ supervised CNN with trainable parameters θ_{sup};
7: $f_{\theta_{unsup}}(x) =$ unsupervised CNN with trainable parameters θ_{unsup};
8: $w(t) =$ unsupervised weight ramp-up function;
9: t in [1, num epochs] ; each minibatch B;
10: $x'_{i\in B} \leftarrow f_{embedding}(x_{i\in B})$ ▷ represent words with word embedding;
11: $z_{i\in B} \leftarrow f_{\theta_{sup}}\left(f_{\theta_{shared}}(x'_{i\in B})\right)$ ▷ evaluate supervised cnn outputs for inputs;
12: $xz'_{i\in B} \leftarrow f_{\theta_{unsup}}\left(f_{\theta_{shared}}(x'_{i\in B})\right)$ ▷ evaluate unsupervised cnn outputs for inputs;
13: $l_{i\in B} \leftarrow -\frac{1}{|B|}\sum_{i\in B\cap S} log\, f_{softmax}(z_i)[y_i]$ ▷ supervised loss component;
14: $l'_{i\in B} \leftarrow \frac{1}{C|B|}\sum_{i\in B}\|z_i - z'_i\|^2$ ▷ unsupervised loss component;
15: $loss \leftarrow l_{i\in B} + w(t) \times l'_{i\in B}$ ▷ total loss;
16: update $\theta_{shared}, \theta_{sup}, \theta_{unsup}$ using, e.g., ADAM ▷ update parameters; θ_{shared}, $\theta_{sup}, \theta_{unsup}$.

As shown in Figure 23, the input x_i is represented as x'_i with the word embedding[3]. Then the representation x'_i is input into three convolutional layers followed by pooling layers to extract low-level features in the shared CNN. Next, we evaluate the network for each input representation x'_i with the supervised path and the unsupervised path to complete two tasks, resulting in prediction vectors z_i and z'_i by concatenating the outputs from pooling layers in these two paths, respectively.

[3] https://www.tensorow.org/api docs/python/tf/nn/embedding_lookup

One task is to learn how to mine patterns of fake news regarding the news labels while the other is to optimize the representations of news without the news labels. Specially, before these two paths, there is a shared CNN to extract low-level features to feed the latter two CNNs. It is similar to deep multi-task learning [225,226] since the low-level features are shared in the different tasks [227,228]. The major difference between the proposed framework and deep multitask learning is that tasks in the proposed framework will involve both supervised learning and unsupervised learning, while all tasks in deep multi-task learning are only based on supervised learning.

In addition, these two paths can have independent CNNs with the identical setups for supervised learning and unsupervised learning, respectively. They generate two prediction vectors that are new representations for the inputs with respect to their tasks. For the identical setups of these two path, i.e., using the same CNN structure for both paths, it is important to notice that, because of dropout regularization, training CNNs in these two paths is a stochastic process. This will result in the two CNNs having different link weights and filters during training. It implies that there will be difference between the two prediction vectors z_i and z_i' of the same input x_i. Given that the original input xi is the same, this difference can be seen as an error and thus minimizing the mean square error (MSE) is a reasonable objective in the learning procedure.

We utilize those two vectors z_i and z_i' to calculate the loss given by

$$Loss = -\frac{1}{|B|} \times \sum_{i \in B \cap S} log\, f_{softmax}(z_i)[y_i] + w(t) \times \frac{1}{C|B|} \times \sum_{i \in B} \|z_i - z_i'\|^2, \quad (3.59)$$

where B is the minibatch in the learning process. The loss consists of two components. As illustrated in Algorithm 7, l_i is the standard cross-entropy loss to evaluate the loss for labeled inputs only. On the other hand, l_i', evaluated for all inputs, penalizes different predictions for the same training input x_i by taking the mean squared error between z_i and z_i'. To combine the supervised loss l_i and unsupervised loss l_i', we scale the latter by time-dependent weighting function $w(t)$ [229] that ramps up, starting from zero, along a Gaussian curve. In the beginning of training, the total loss and the learning gradients are dominated by the supervised loss component, i.e., the labeled data only. Unlabeled data will contribute more than the labeled data at later stage of training.

In the test phase, fake news detection has been completed by classifying the news into True class or False class. The proposed model has two paths with supervised CNN and unsupervised CNN, where the supervised CNN is able to classify the news. In other words, we only use the supervised CNN to accomplish fake news detection in the test phase.

Although there are a few related works in the literature such as the Π model [229], there exist significant differences. Laine *et al*. [229] proposed Π model by introducing self-ensembling in the learning procedure to address the lacking of labeled images for image classification. It is to learn high-performance deep learning models on large image datasets, where only small portion of the datasets contains image labels. The Π model has two branches to learn on labeled images and unlabeled images for classifying images and enhancing image representations, respectively. Specifically, the Π model employed image augmentation to enhance the model performance, where the augmentation procedure will involve image manipulations such as cropping images and rotating images to enlarge the training datasets. As two-dimension data, pixels in images have spatial correlations to implement image augmentation. However, it is very different for text data and it is not clear how to enlarge text training datasets with these manipulations since text data does not have this type of spatial correlations for data augmentation. Therefore, comparing performances between Π model and the proposed model directly will not be appropriate since we cannot directly use Π model for text data such as in fake news detection. On the contrary, the proposed model will not rely on data augmentation to enhance model performance. In addition, instead of using one path CNN, the proposed model will be more flexible as the two independent paths in the proposed model can be tuned for specific goals.

3.4 Implementation, Visualization, and Validation

The implementation of real world applications using data analytics is explored in this research thrust to validate the theoretical approaches. In addition, visualization platforms and impact of high performance computing are also investigated.

3.4.1 State-of-The-Art

3.4.1.1 Implementation of real world applications in UAV tracking

Since AlexNet [230] won ImageNet in 2012, cnn has been widely used in the computer vision. This is because cnn can dramatically reduce the number of parameters with parameter sharing and

better ability of extracting the visual features. Through the multiple convolutional layers, the fine and simple features can be integrated into the expressive and complicated features. For instance, the eyes could be composed of the edge, color and texture. In this section, we are focused on the fundamental concepts that are used in the object detection and object tracking as well as the techniques usually applied for the cnn.

Exploiting AlexNet, rcnn [231] takes advantages of its superior ability of the classification by proposing a lot of regions in the image and classifying them. The region proposals are also considered as the bounding boxes. Basically, rcnn integrates the convolutional part of AlexNet, which can generate the feature maps, into the selective search [232]. On the other hand, it uses svm and linear regression for determining the class of the object and localizing its corresponding bounding box respectively. However, rcnn is composed of three models and each of them has to be trained separately. In addition, it takes time to process every bounding box, where the total number of the bounding boxes would be about 2000 per image. Adopting multi-task learning, Fast rcnn, an end-to-end model (i.e. a single cnn), replaces the svm and separate linear regression with the softmax classifier and linear regression both of which share the same layer: roi. The purpose of roi is used for speeding up both the training and inference in rcnn by reusing the feature maps so that only one pass of the image is applied to rcnn instead of many passes of the many regions. Another improved version of rcnn|Faster rcnn|has been proposed in [233] to solve the problem of creating a lot of bounding boxes, which in turn has a serious impact on the processing speed. Faster rcnn reuses the same feature maps produced by cnn to create the bounding boxes rather than the selective search. The feature maps are processed by a cnn known as the Region Proposal Network with the sliding windows.

YOLO [234] also utilizes AlexNet but is focused on the speed of the inference. Other than rcnn that classifies the proposed bounding boxes to perform object detection, it regards object detection as the regression problem to spatially separated bounding boxes and associated class probability. Besides, it divides an image into a grid, where a cell is in charge of detecting the object if the center of the object is located in itself. The base YOLO model can achieve 45 fps while processing the 448x448 images. In [235], several improvements of YOLO are proposed: batch normalization, convolutional with anchor boxes, multi-scale training etc.

Having to localize the objects initially, object tracking usually exploit the temporal information of the objects from the previous frame or image. Conventionally, similarity measure is conceptually

and intuitively used to find the most likely objects (i.e. translations). With the similarity measure, we can find the smallest difference between the target specified initially and the object which we search in a specific region. A cnn for object tracking with similarity measure is presented in [236]. The authors call it siamese networks since it has two convolutional parts, where one is used for processing the target specified initially and the other processes the search image to propose the possible objects. At the final stage, a scalar-value score map plays the similarity function to determine which one is the most likely object. The size of the search image is double the size of the target at default. The improved version of the siamese networks is proposed in [237]. It uses the Correlation Filter, which has the closed-form solution as a differentiable layer, to learn a linear template. With the help of the Correlation Filter, the searching of the most likely object` can be enhanced.

We discuss the area of uav object-following algorithms first and then the area of deep learning on embedded platform. Two main methods are used in uav object-following algorithms. The first one is nonlinear robust adaptive tracking control [238]. Using the robust integral of the rise method and an iandi-based adaptive control method, a novel asymptotic tracking controller can be developed. The rise technique is applied in the uav attitude control for disturbance rejection, whereas the iandi approach is chosen for the uav position control to compensate for the parametric uncertainties. The second method is inverse reinforcement learning based tracking control [239]. In this approach, inverse reinforcement learning has been used to learn the cost function from the historical test flights firstly. Then, a reinforcement learning based tracking control has been designed to minimize the learnt cost function as well as tracking errors. Traditionally, path planning and flight control are designed offline and programmed onboard. Currently, through adopting reinforcement learning along with adaptive control, the online path planning and flight control can be done if the onboard microprocessor has enough computation ability, which causes the difficulty of real-time tracking on the embedded systems.

Thus, deep learning on embedded platform becomes the potential approach recently. In [240], the authors presented an exploration of different single-shot cnn detectors for uav-based vehicle detection. The resulting cnn referred as DroNet is capable of performing 18 fps at most. The further improved version of DroNet is proposed in [241]. It is focused the detection only on promising image regions and achieved 20 fps on in a CPU platform, which is a laptop platform with an i5-8250U CPU and 8GB RAM. To detect the small target, [242] proposed an early visual attention

mechanism, called rcn, to choose the most promising regions with small objects and their context. rcn allows to work with high resolution feature maps but with a reduced memory usage. However, in our scenario, the size of target varies and could be significantly large when the target is close to uav. Another work for real-time object detection is proposed in [243] by introducing Tiny ssd. Tiny ssd is composed of a non-uniform highly optimized Fire subnetwork stack, which feeds into a non-uniform sub-network stack of highly optimized ssd-based auxiliary convolutional feature layers, designed specifically to minimize model size while retaining object detection performance. The authors claimed the Tiny ssd is 26X smaller than Tiny YOLO but did show any number about fps and which embedded platform they run Tiny ssd.

We experimented Tiny YOLO and CFNet [237] on the NVIDIA Jetson TX2 with the same dataset used in the paper. Written in C and CUDA, Tiny YOLO could achieve 19 fps. With TensorFlow, CFNet achieved 7 fps. Both of them are used to process the images in size of 360x640.

3.4.1.2 Data Visualization on cloud

A wealth of visualization techniques have been applied to a variety of massive military datasets in various military application domains, including Command and Control, Intelligence, Logistics and Information Operations, and Decision making [244].

Visualization is an effective approach to visually represent data and communicate information-clearly and effectively through graphical means and user-system interaction. It helps us look into a rather sparse and complex data set by communicating its key-aspects in a more intuitive way, and find patterns and trends that are otherwise hidden and turning complex data into actionable insight. Visualization approaches can be characterized in terms of visual representations used (e.g., graphs, charts, maps); visual enhancements (e.g., use of overlays, distortion, animation); interaction (e.g., direct manipulation, drag and drop, haptic techniques); and deployment, which includes the computing environment (display devices, software) and advanced deployment techniques (such as intelligent user support and enterprise integration).

Choo *et.al*. [245] discussed the interplay between precision and convergence. The authors presented customizing computational methods to include low-precision computation and iteration-level visualizations to ensure real-time visual analytics for big data. A new kind of visualization space called hybrid-reality environment is proposed in [246] to achieve scalable real-time visualization of heterogeneous datasets. The environment is able to synergize the capabilities of virtual

reality and high-resolution tiled LCD walls, allowing users to juxtapose 2D and 3D datasets and create hybrid 2D-3D information spaces. The authors presented two such environments and Cyber-Commons and CAVE2, which are hybrid-reality environments that provide high-resolution stereoscopic display surfaces, creating hybrid-reality spaces that blur the line between virtual environment and tiled display walls. Zhang and Huang [247] proposed the 5Ws model by using 5Ws data dimension for big data analysis and visualization. A visual-analytics platform named DIVE (Data Intensive Visualization Engine) is presented in [248], which is a data-agnostic, ontologically expressive software framework that is capable of streaming large datasets at interactive speeds. DIVE employs structured-data-model manipulation strategy to process high-throughput streaming of large structured datasets. Biem *et.al*. [249] proposed a Streaming Time-Series Analysis and Management (STAM) system to easily manage, analyze, and visualize large multidimensional time series, with dimensions on the order of hundreds of thousands. Gouin [244] conducted a thorough survey on the existing visualization techniques and approaches adopted by military operations, and presented their C3I (Command, Control, Communications and intelligence) knowledge-based visualization system. Very few technical articles focusing on military visualization applications are publicly available.

3.4.2 Motivation

3.4.2.1 Implementation of real world applications in UAV tracking

As cnn reshapes computer vision in terms of Deep Learning, the advancement of object detection and tracking has gained a huge success as the structure of cnn goes deeper, which in turn achieves higher accuracy. However, all these good things come from the complicated models and the underlying high-end computing resources such as gpu. Thus, the applications or services associated with cnn have to be run or provided by the data centers or clusters through the Internet. It is not practical for many applications to rely on the assumptions of the accessibility of the high-end computing resources and the Internet, especially for those real-time applications running on embedded systems with a poor accessibility of the Internet.

As illustrated in Figure 25, object detection and tracking on the uav is such a case we just describe above. The uav, equipped with a camera, a battery and an embedded system (a mobile GPU), must navigate itself based on the information provided by the module of object detection and tracking (a deep learning model) running on the embedded system.

Figure 25: Self-navigating uav for single object detection.

There exist many challenges when designing such a deep learning model:

- Timely response is required
- Poor or no accessibility of the Internet
- Limited computing resources on board such as an embedded system
- Flexibility of customization is needed to meet the special requirements due to the characteristics of the tasks

In addition, the problems are also common to those in mobile edge computing and iot devices if we deploy deep learning applications on them. There are two main approaches to address the problems: (1) compress a trained deep learning model, and (2) train a relatively small model. Examples of model compression include: optimization of the convolution or operations [250, 251], quantization of the parameters [252, 253], and simplification of the model structure [254{257]. These approaches assume that there already exists a pre-trained model and compressing it would not sacrifice the accuracy a lot but would speed up the inference. However, most of these well-trained models have the tendency of being used for the applications of general purpose. For example,

AlexNet [230] is able to classify 1,000 classes in ImageNet with very good accuracy (about 37.5% top-1 error rate). However, in the case of object detection and tracking on the UAV considered in this work, we may need to classify the objects of a new class or a compound object comprising several objects of different classes. Even though the pre-trained model could be used as a feature extractor, the resulted model would be an overkill even with fine tuning. As a result, additional effort is needed to handle these problems that would consume the time saved by model compression, and it will sacrifice the speed of inference which is a major concern for real-time applications. Thus, we set out to develop a modeling method to train a relatively small model that integrates the important concepts drawn from object detection and tracking. Furthermore, transferring the weights of some specific layers to reuse the low-level features could reduce the overhead of transfer learning and might be helpful for the model development.

3.4.2.2 Data Visualization on cloud

The massive data collected from battle-fields can be in various formats, such as text, voice, picture, video, and environmental data, which make visualization challenging to present all of them friendly and intuitively. Moreover, military operations require real-time information presentation from the constant streaming data, and high interactivity to respond events and make tactical and strategic control and commands. In our previous work [258], we developed new methods using Self-Organizing Map (SOM), Principal Components Analysis (PCA), and 3D data plotting to discover gene patterns, reduce gene data dimensions, and visualize the final results interactively in 3D space. A new approach using partial distortion method to approximate the water droplets and visual effects caused by water droplets on glass is presented in [259]. New methods to perform nonlinear perspective projections and 3D view deformation are proposed in [260], and we also developed subdivision-based 3D surface modeling [261] as demonstrated in Fig.26 for F-16 fighter jet.

Given the targeted military dataset, in this research thrust we will explore the existing visualization technology and study the feature, attributes, and size of the targeted dataset, and invent novel visualization approach such as virtual reality based methods to perform real-time visualization of massive military datasets including video streams. We will move the visualization into the cloud system, design and implement a cloud-based interactive visualization interface to present battle-field information in real-time. The interface will be rich-content to include multimedia, animation,

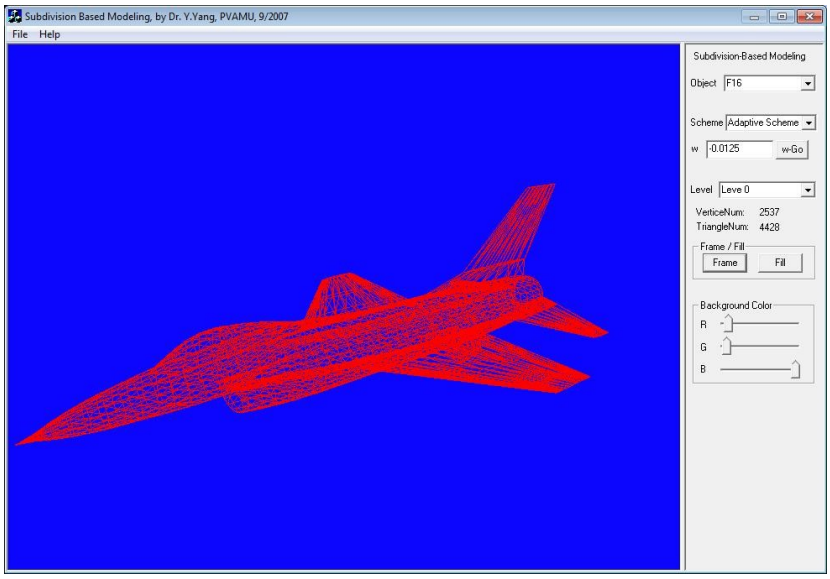
maps, graph, charts, and be highly interactive. Research will focus on interactive information visualization, including collaboration, portal technology, synchronized views, immersive displays, abstract representation, and 3D territorial models.

3.4.3 Problem Formulation and Proposed Approach

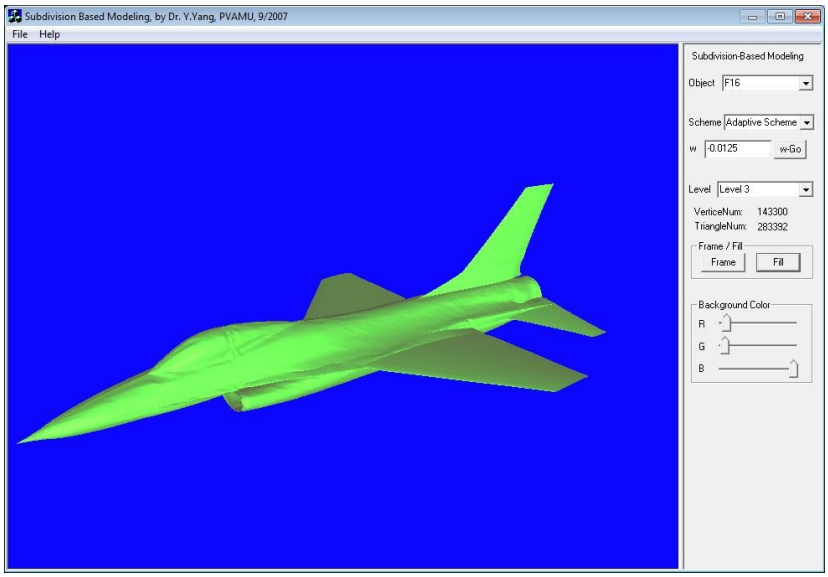
3.4.3.1 Implementation of real world applications in UAV tracking

The demand for mission critical tasks, especially for tracking on the UAVs, has been increasing due to their superior mobility. Out of necessity, the ability of processing large images emerges for object detection or tracking with UAVs. As such, the requirements of low latency and lack of Internet access under some circumstances become the major challenges.

In this work, a novel modeling method is proposed to provide insights of developing a small model suitable for embedded systems running a real-time application, specifically, a self-navigating uav for single object detection and tracking. We identify the special requirements of this task and design the model architecture by taking advantage of the concepts drawn from YOLO [262] and rcnn [231]. In addition, we propose a novel "inducing neural network" as part of the model architecture which is not for classification and localization but helps to speed up the training.



(a) F-16 level-0 model: 2537 vertices and 4428 triangles



(b)F-16 level-1 model: 9494 vertices and 17712 triangles

Figure 26: Subdivision-based 3D surface modeling

We train the model from scratch with only the given object datasets (DAC 2018 datasets [263]) to assess the capability of the proposed method. We also experiment with NVIDIA Jetson TX2 and demonstrate that our trained model can achieve more than 20 fps, thus fulfill the real-time requirement. It is shown in field tests that the uav with our design model running on NVIDIA Jetson TX2

can use the information provided by the model to navigate itself to follow the target and work together with other sensors onboard the UAV.

Scenario: The general mission of the self-navigating uav is following the single object. The object could be anything specific like a blue car or a person walking with the other people. The embedded system must be able to process the image taken from the camera on the uav, localize the object in the image including the determination of whether the object is outside of the view or not, and control the uav to follow the object or stay around using the localizing information. Not being able to rely on the Internet to send the images to the high-end servers that can accommodate the complicated model and provide the fast inference, we suppose that the access of the Internet is limited and only the commands of controlling the uavs are allowed. Although the embedded system is required to be capable of tracking the object in real-time, which is vague and would vary as the applications, we assume that the uav might lose the object due to the processing time or the strong winds so the model for the single object detection is chosen to develop. This is because the object might appear easily outside of the searching region used for the object tracking, and enlarging the searching region could cause the performance degradation. In this section, we present our modeling method with the dataset provided by DAC 2018 [263]. It includes 98 classes and each contains about 1000 images. The image size is 360x640 and only one object needs to be tracked. In addition, this dataset guarantees that there must be one object in every image. Figure 27 shows four samples and one of them asks to track the person riding on the bike. It also implies that the object we are interested in could be a compound object that is composed of two known objects. Furthermore, the object could be probably very small on the scale of the image: the boat for example.

Structure Choice: To design the structure of our cnn, we adopt the explanation of understanding cnn in [264]. The structure of our model would be composed of two parts: feature extraction and semantic interpretation. The feature extraction is used to extract the necessary features and generate the feature maps so that the semantic interpretation can make use of them to localize the object. Figure 28 illustrates the structure of our model that can be mainly divided into feature extraction and semantic interpretation. The pattern of the structure is based on the following: $Input \longrightarrow [CONV \longrightarrow RELU \longrightarrow CONV \longrightarrow RELU \longrightarrow POOL] \times 3 \longrightarrow [FC \longrightarrow RELU] \times 2 \longrightarrow FC$ [265]. Under this structure, we are focused on how to utilize it in an efficient way.

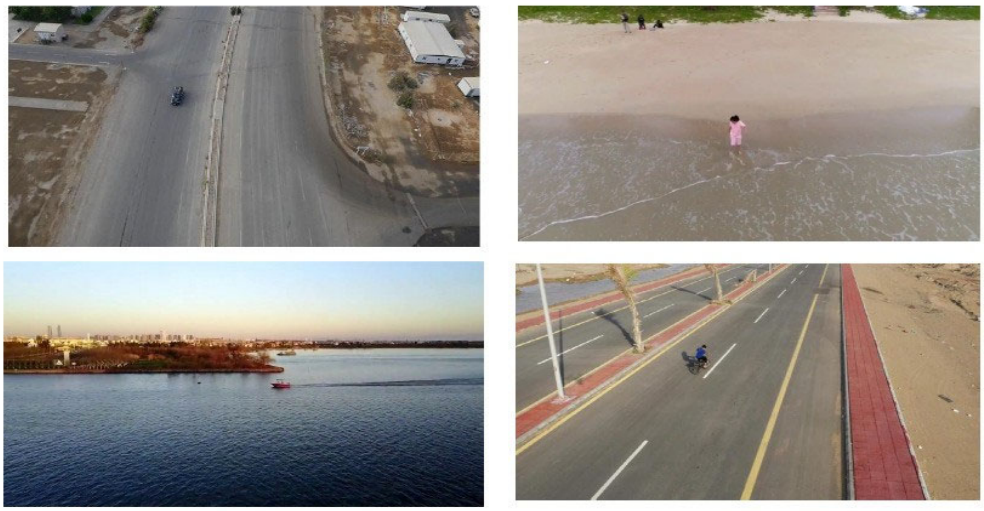


Figure 27: Four samples of the dataset of DAC 2018. From top to bottom and left to right, the object is car, child, boat and person riding on a bike, respectively.

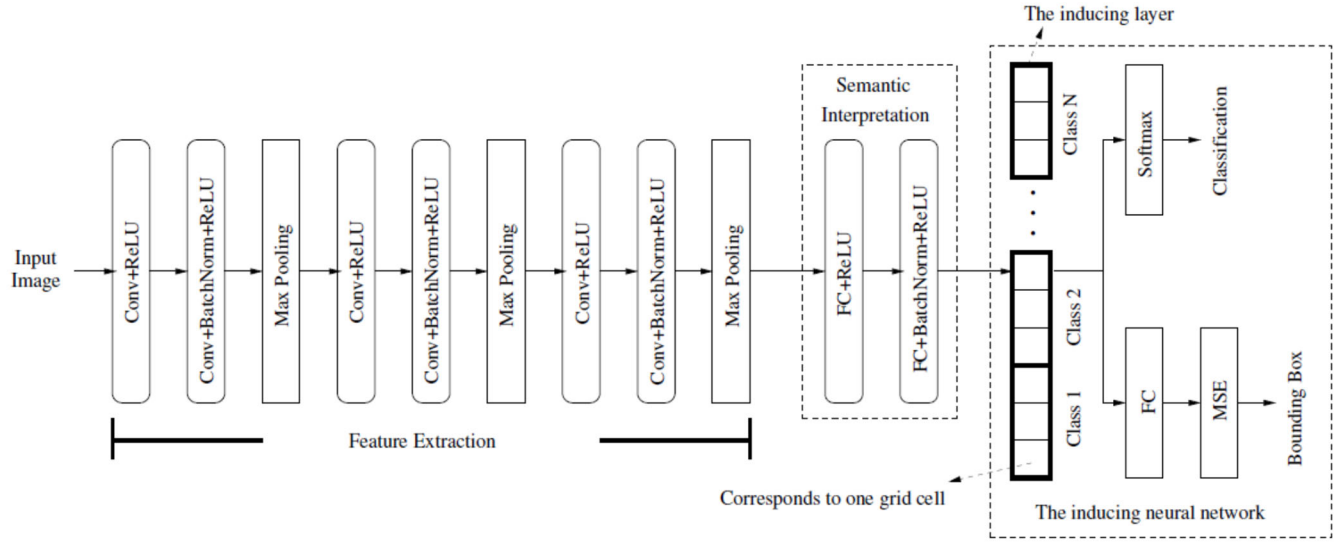


Figure 28: Architecture of our proposed cnn.

As to the mission of tracking on the uav, the images are often in the high resolution to make sure that the object is able to be recognized. Thus, it is not practical to resize the images into the smaller ones since we would probably lose the detail information of the object. At the first CONV layer, we set the strides of these filters larger in comparison with the other CONV layers. With the larger

strides, we can reduce the images through applying such the filters to them. To compensate the information lost by using the larger strides, we choose the larger filter size CONV and imply that the fine details are not what we care about. If the object is the person riding on the bike for example, we don't care about which part is that person and which part is the bike. Nevertheless, we need the details of the features of this object such as the legs on the pedals instead of the features of the legs and the pedals separately.

The number of the filters in CONV layers and the neurons in FC is determined by the trial and error. That is, we have to experiment with the model running on the embedded systems to see if it can satisfy the real-time requirement without sacrificing the accuracy too much. Table 5 shows the specifications of the CONV and FC layers before the inducing neural network. Note that the stride of the first CONV is 2x3 which would reduce the image size at the very beginning.

Table 5: The specifications of CONV and FC before the inducing neural network.

Layer	Filter size	Number of filters or neurons	Stride
CONV	6x10	32	2x3
CONV	3x3	32	1x1
MAXPOOL	2x2		2x2
CONV	3x3	64	1x1
CONV	3x3	64	1x1
MAXPOOL	2x2		2x2
CONV	3x3	128	1x1
CONV	3x3	128	1x1
MAXPOOL	2x2		2x2
FC		1024	
FC		1024	

Inducing Neural Network: Conventionally the model constructed according to Table 5 can be trained directly to predict the bounding box. However, it is very hard for us to make the model converge. We reckon that the traditional learning scheme, relying on only the loss function of regression, has the difficulty of searching the good direction of optimizing the model in the reasonable time. Here we use mse as the loss function of regression. As a result, we adopt the multi-task learning used in rcnn| classifying and detecting the object together. However, it also performs not quite well so we drew the concept of dividing the image into a grid from YOLO and integrate

it into the multi-task learning. In our learning scheme, there is an implicit grid that we use to guide the backpropagation. The intuition is that we want the procedure of the optimization smarter. This grid is represented by one FC (i.e. an 1-D array) and the length is N × G, where N is the number of the classes and G is the grid size. For instance, there are 98 classes and we divide an image into 512 grid cells. N is 98 and G is 512. This is the essential part of the inducing neural network since we plan to use it to update the parameters in a controlled way instead of random guess. We call it the inducing layer. As shown in Figure 29, we use a matrix that is not trainable to multiply the-inducing layer and the output is used for the softmax classifier. Since each cell in the output corresponds to only one class, we can optimize the corresponding parts smartly. For example, if the ground truth is 2 but the model predicts 1, the optimization would be backpropagated through the ones in the matrix (i.e., the neural parts of 1 and 2), and the zeros would block the irrelevant optimization like 3.

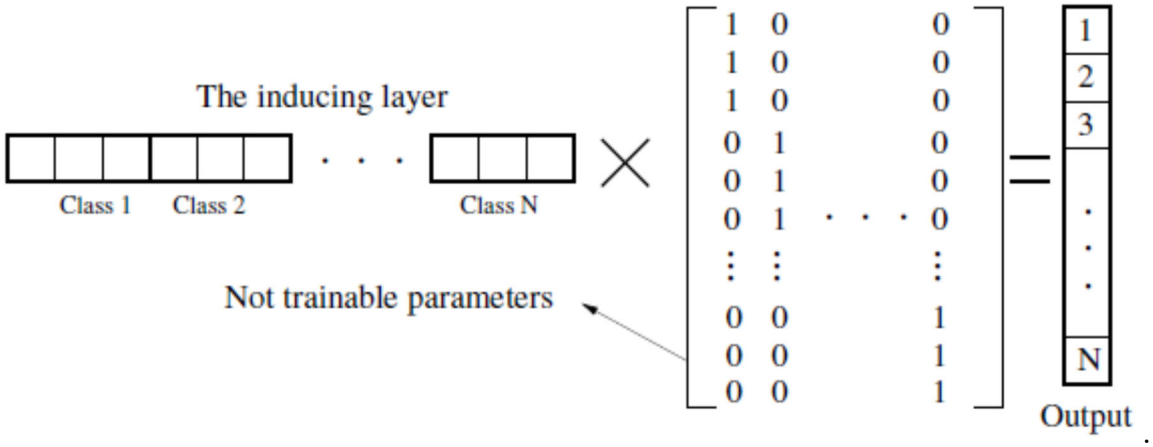


Figure 29: Illustration of how the inducing layer works smartly while optimizing the model

It is straightforward for the softmax classifer to use such this matrix, whereas it cannot be applied to the loss function of regression easily. As shown in Figure 30, we use two matrices to construct the similar matrix in Figure 29 implicitly. N, G, and P represent the number of the classes, the grid size, and the number of the values needed for the bounding boxes respectively. In the first matrix, we think of the sub-matrix as the part which is in charge of the single object detection of one class. Similar to Figure 29 and regarding the sub-matrix as the unit, only one unit is 1 and the others are 0 in each column. The size of the sub-matrix is $G \times P$ which implies that it would use the gird to predict the bounding box for one class. There are N classes so the size of

the first matrix is $N * G \times N * P$. Then, the second matrix maps these $N * P$ values to P. Note that the first matrix is not trainable either.

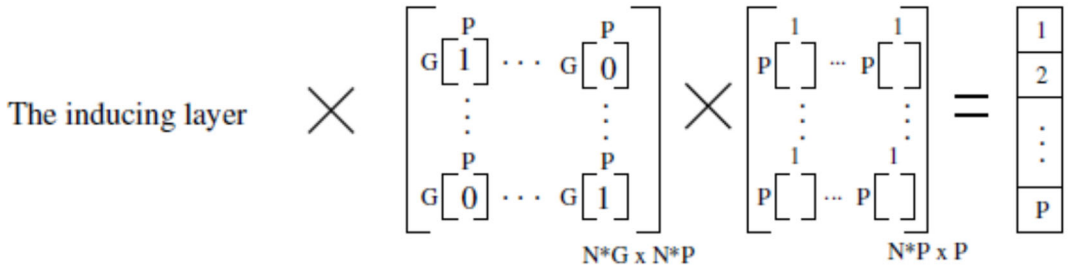


Figure 30: Illustration of how the inducing layer works smartly while optimizing the model

Settings of Hyperparameters: MomentumOptimizer is used for training and the loss function is shown as below:

$$loss = w_m \times MSN + w_c \times softmax + regularization_{L2}$$

Since we train the model from the scratch with the multi-task learning, it is natural to see that the training curves of both the softmax and mse pull and push each other. On the other hand, we don't want either of them dominates the other so that making progress of the other is poor or prohibited. The empirical weights of the softmax and mse we used in our model are 1e-2 (w_m) and 1e-3 (w_c) respectively. We assign the heavier weight to the softmax to lead the direction of the optimization because the scenarios of those 98 classes are quite different from each other. Classifying the image first would help our model to detect the object more easily, which means that the convergence of our model can be achieved in a reasonable time. Note that the ratio of softmax to mse depends on the characteristics of the datasets. The good ratio is supposed to have the good classification accuracy and make the model converge smoothly. Nevertheless, the too strong emphasis on the softmax would have a negative effect on the mse, which in turn lowers the iou.

The learning rate is set to 1e-3 initially. We changed it to 1e-4 and 1e-6 at the 20th and 60th Epoch respectively to avoid the dramatic oscillation. We find that the validation accuracy and iou become jittery and the deviations of them are high after 100 Epochs. Thus, we evaluate our model at every

Epoch and stop training at the 100th Epoch. During this training, we pick the best result as our trained model.

Before feeding the images to the model, we subtract every pixel of the images from the corresponding mean value in that pixel. To speed up the convergence of our model, we apply the batch normalization (**BatchNorm**) every other layer. We initialize the weights with the Xavier initializing method and the grid size is 512.

Analysis: We select the 10,000 images from the dataset randomly as our validation dataset, whereas the remaining dataset is considered as our training dataset (86,408). We use two NVIDIA Tesla P100 gpus to train our model with the batch size 128. With the multi-task learning, we evaluate our model in terms of the accuracy of classification and iou for the object detection during the training. Figure 31 shows the validation accuracy and iou for the classification and object detection respectively. As what you can see, it is a proof that our modeling method is able to associate the prediction of the bounding box with the classification. However, our model still suffers from a sudden drop after the 100 Epochs, and makes no progress for the model. It takes about 15 hours to train our model in 100 Epochs and our model can achieve about 50% iou with more than 90% classification accuracy at the 20th Epoch. That is, it would take 3 hours to have a feasible model which is trained from the scratch. Among the 100 Epochs, the iou and classification accuracy of our best model are 69.37% and 99.81%.

Comparing Figure 32(a) and Figure 32(b), we find out that the training curve of Figure 32(a) is more stable and faster to perform more accurate classification and higher iou than the one of Figure 32(b) before the 50th Epoch. Furthermore, in Figure 32(b), there is a early sudden drop in the accuracy of the classification as well as iou. We think of such sudden drop as the indication that our model arrives at the local optimum and it is difficult to improve our model further. In Figure 32(b), the iou and classification accuracy of the best model are 64.54% and 99.61%. Besides, the model converges faster and more smoothly with the mean subtraction before the 50th Epoch so the following experiments would include such preprocessing.

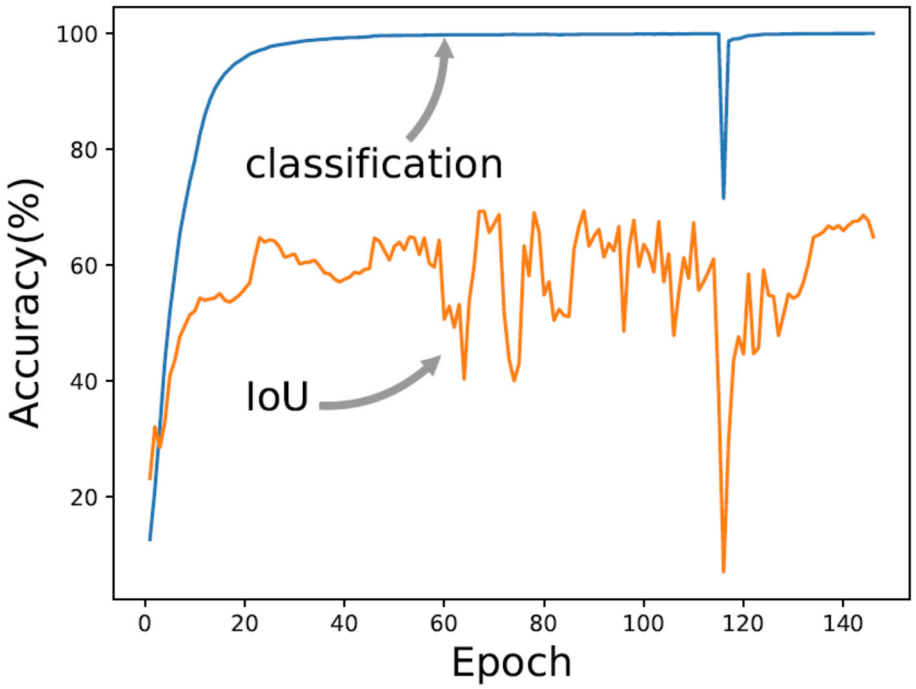


Figure 31: The validation accuracy and iou of our model with the grid size 512 during the Training.

Figures 33(a) and Figure 33(b) show the training curves in the different grid sizes. In Figure 33(a), the classification accuracy converges slower than the one in Figure 31, whereas the classification accuracy in Figure 33(b) converges faster than the one in Figure 31. On the other hand, we find that the training curve of iou in Figure 33(b) is more stable in the larger grid size. As such, we conclude that the larger the grid size, the better the classification accuracy and the more stable the iou. However, both the ious of the models with the grid size 256 and 1024 converge slower than the one with the grid size 512. We reckon that our learning rates are small so the model is trapped in the local optimum. Even though we increase the learning rate to see if we can get the better model, the result is not satisfying. Thus, we choose 512 as our default grid size for this dataset.

To what extent can the inducing neural network affect the model? In Figure 34(a), we remove the contribution made by the softmax from the optimization. Regardless of the classification, the iou converges in a reasonable rate that we expect. Interestingly, the classification accuracy is always less than 1% but the fluctuation of iou still exists where we think it should have been alleviated. Replacing the inducing neural network with the equivalent FC in Figure 34(b), we find

that the training curve of the iou is not stable but the classification accuracy is a little better than Figure 34(a). As a result, we conclude that the inducing neural network helps to stabilize and accelerate the training of the model in the early phase, whereas contributes less to the model improvement in the late phase. We consider the enhancement of training the model in the late phase as the future works. This is because about 50% of the iou is good enough for us and the real-time matters more under our scenario. To achieve more than 50% of iou, the other methods such as non-maximum suppression [266] should be included.

Replacing the inducing neural network with the equivalent fully connected layers, Figure 35(b) shows that the iou fluctuates more heavily than Figure 35(a). Also, it proves that our modeling method has the better ability of improving the iou in the early phase. Namely, developing the model with our modeling method is faster than the conventional way.

3.4.3.2 Data Visualization on cloud

In our preliminary efforts [267], we built a big data analytics cloud platform with special interests in geophysics data sets. The cloud platform is named Seismic Analytics Cloud (SAC) to process and analyze seismic data with the deep learning capacity. Although the platform is able to process and analyze big seismic data sets with scalable performance, the big data visualization on the cloud remains a challenge to us. In this project, we aim to improve the visualization efficiency of the SAC platform while users conduct the seismic data analytics. At the beginning, SAC was mainly developed as a new computing platform with a balance of both performance and productivity, and featured with big data analytics capability. Soon afterward, we collaborated with Thermo Fisher Scientific to integrate the rendering of seismic slices in SAC platform. The platform can now support the management of seismic data volumes, attributes processing, seismic analytics model development, workflow execution, and 3D volume visualization on a scalable, distributed computing platform. However, for a big 3D seismic volume takes a long time (over a minute) to display a rendered 3D image on the platform via a web browser. To address this challenge, we collaborated with Thermo Fisher Scientific to bring the high quality and high-performance rendering of seismic volume in SAC using Open Inventor (https://www.openinventor.com).

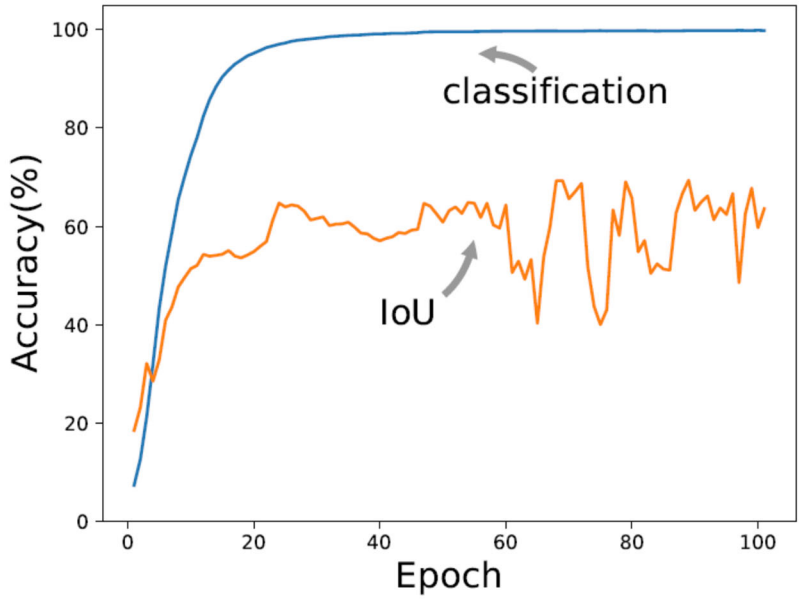
The objective of SAC is to deliver a scalable and domain-specific cloud platform to facilitate the seismic data analytics research and development in the geophysical areas such as the oil/gas exploration or the earthquake detection. Figure 36 shows the overall system structure used in SAC.

The bottom layer is operating systems SAC can build on; next layer is to provide the JAVA and Python runtimes with Hadoop Distributed File System (HDFS for storing the large seismic data files and No-SQL database Cassandra for metadata and data attributes. Standalone, Mesos, and YARN are all supported on our platform for the resource management. We use Apache Spark as the big data parallel processing engine together with the widely-used signal and image processing libraries to provide scalable performance and good productivity. At the very top layer is SAC SDK, workflow, templates, and visualization module. Researchers can build their own seismic data analytics work on top of the cloud platform.

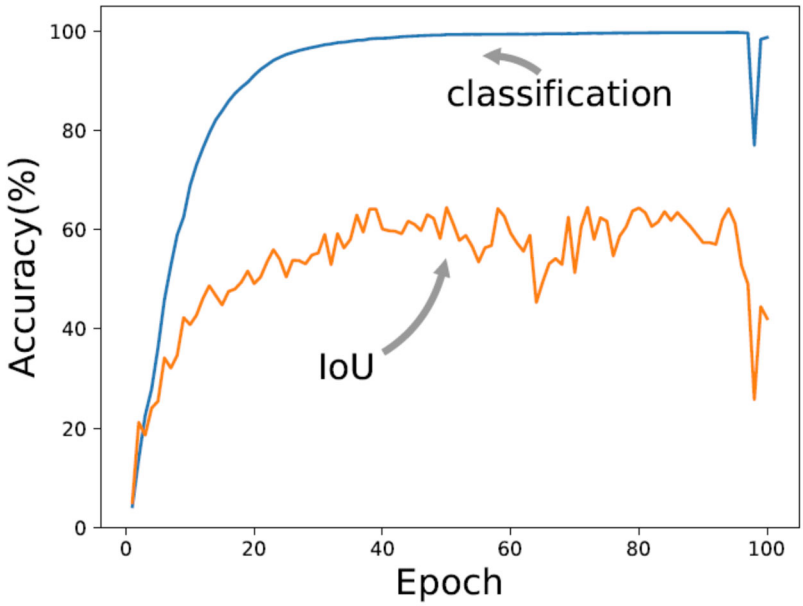
To efficiently process the large seismic data, we need to build SAC on a scalable computing platform. We chose Apache Hadoop, a collection of open-source software utilities, and Apache Spark, an open-source distributed general-purpose cluster-computing framework, for processing seismic data. These two frameworks are widely used for big data analytics as they partition the data into small chunks and distribute these chunks across worker nodes to achieve scalable performance.

Apache Spark and HDFS

Apache Spark is an open-source distributed cluster computing framework developed by AMPLab at the University of California, Berkeley. Compared to the MapReduce technique in Hadoop, Spark provides a resilient distributed dataset (RDD) that keeps data processing in memory to reduce the data IO. To use Spark efficiently, it requires a cluster resource manager and a distributed file system. Spark supports standalone (native Spark cluster), Hadoop YARN, or Apache Mesos. For the distributed storage system, it provides an interface for Hadoop Distributed File System (HDFS), Cassandra, OpenStack Swift, Amazon S3, or a custom solution. We use HDFS for the distributed fault-tolerant file system in SAC.

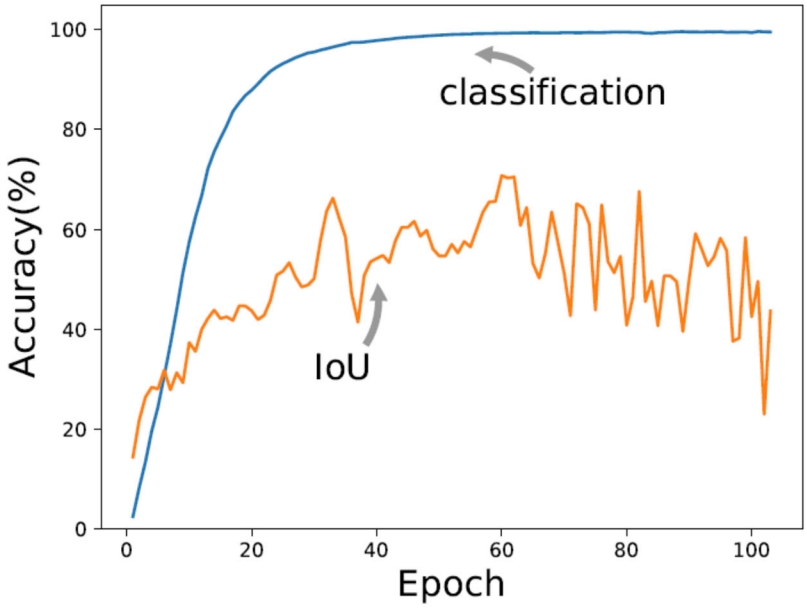


(a) Subtracting the mean during training.

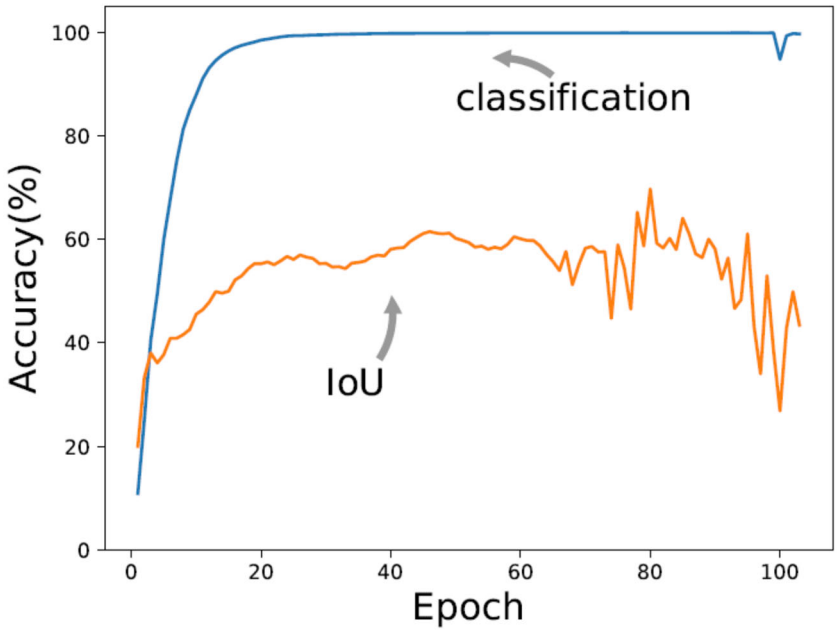


(b) Without subtracting the mean during training.

Figure 32: The impact of subtracting the mean while training the model.

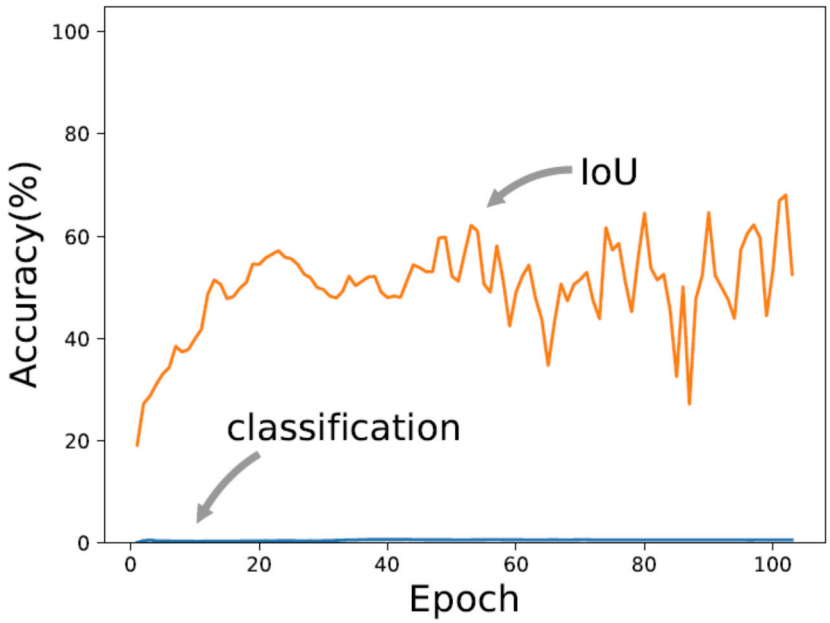


(a) The grids size is 256.

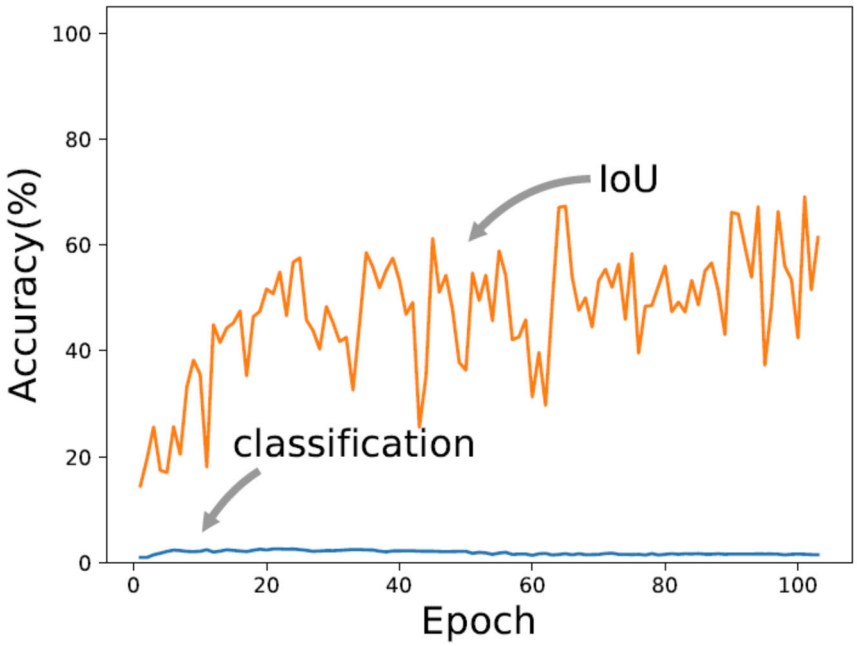


(b) The grid size is 1024.

Figure 33: The impact of the grid size while training the model.

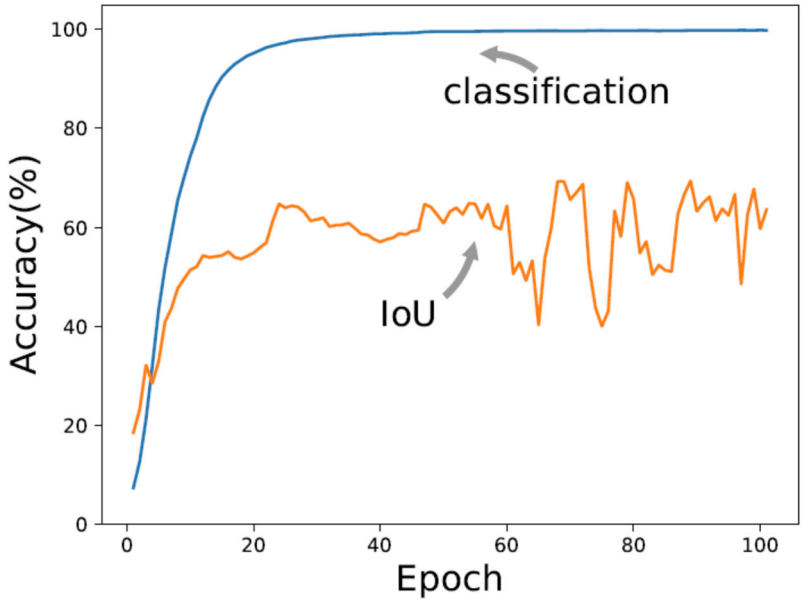


(a) mse with only the inducing neural network.

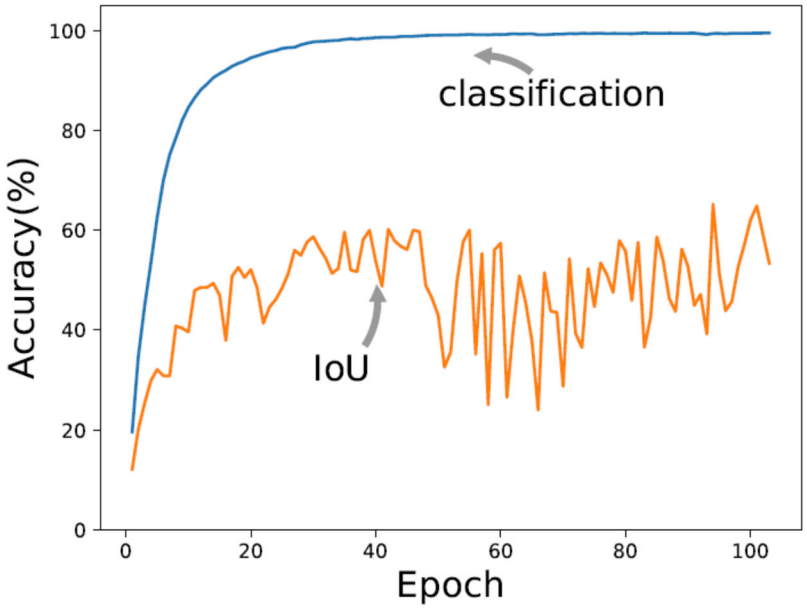


(b) mse without the inducing neural network.

Figure 34: The impact of the inducing neural network on mse.



(a) mse with only the inducing neural network.



(b) mse without the inducing neural network.

Figure 35: The impact of the inducing neural network on the training.

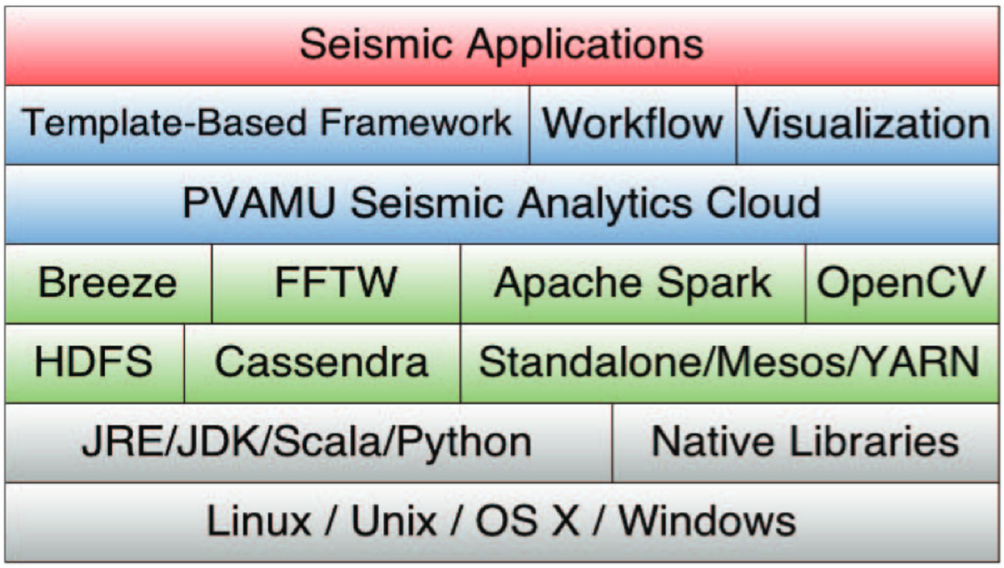


Figure 36: The Software Stack of Seismic Analytics Cloud Platform

Communication

In this project, we use ZeroMQ (https://zeromq.org/) to transfer messages among the data service and rendering service. ZeroMQ allocates sockets to users to load their message and transfer the message across different types of transportation like inprocess, inter-process, TCP, and multicast. The advantage of ZeroMQ is that it can swiftly transfer messages in a cluster. Its asynchronous I/O model allow users building a scalable multicore application.

We use ProtoBuf (https://developers.google.com/protocol-buffers) to define the communication protocol between these services. ProtoBuf is a flexible, efficient, automated mechanism for serializing structured data. Users can define the communication protocol and data structures themselves. Then, they can use the ProtoBuf generated source code to read and write that specific structured data from a variety of data streams. When there is something new to update, programmers can directly update the data protocol and data structures without changing the programs. Also, it can be used in many language codes, includes Java, C++, and Python.

OpenInventor

Open Inventor is a 3d-visualization toolkit offering a high-level object-oriented graphics libraries (API) for creating advanced 3d-visualization applications [9]. It provides a set of dedicated extensions for developing visualization solution for various data types, such as geometries, volume, mesh, and images, and for implementing remote/cloud-based rendering capabilities. In addition to hardware-accelerated volume rendering, VolumeViz extension of OpenInventor provides a large data management (LDM) technology to manage out-of-core loading of large volumetric data that do not fit in the available system memory and far exceeds the video memory capacity even on high-end graphics cards. Since Open Inventor uses GPU for volume rendering, it can only render seismic data that fits in the limited video memory. To address this limitation, LDM creates a hierarchical, multi-resolution bricked representation of seismic volume to allow out-of-core loading of data as needed for rendering.

Open Inventor avoids loading whole seismic volume in full-resolution, instead only loads bricks from different resolution levels depending upon the camera position and available system and video memory. Another advantage of creating hierarchical representation of seismic volume is that it allows Open Inventor for quick rendering of volume in low resolution, and progressively render to higher resolution, without impacting interactive quality of the visualization tool.

Slice rendering with a set of bricks from different resolution levels is shown in Figure 37. The bright green boxes represent full-resolution bricks that are closer to the camera; and dark green boxes represent low resolutions bricks for volume region away from the camera for available memory.

The bricked representation of seismic volume also makes it possible to store bricks (chunks of volume) in a distributed file system in the cluster. Open Inventor will request data server for individual bricks, as needed, for the volume rendering.

Open Inventor also provides remote visualization technology, RemoteViz, that makes it possible to perform remote rendering of large data in a dedicated high capacity GPUenabled server node. The rendered images are transmitted over web to a web application in a clients devices, which could be a tablet, a smart phone, a laptop or a desktop. PVAMUs SAC platform uses RemoteViz technology to render seismic volume on a remote rendering server and display the rendered image on the web-page to be accessible from anywhere. RemoteViz also manages 3D interaction from

users, performs bandwidth calibration to adjust quality vs interactivity, and supports VP9 and H.264 encoders for streaming images.

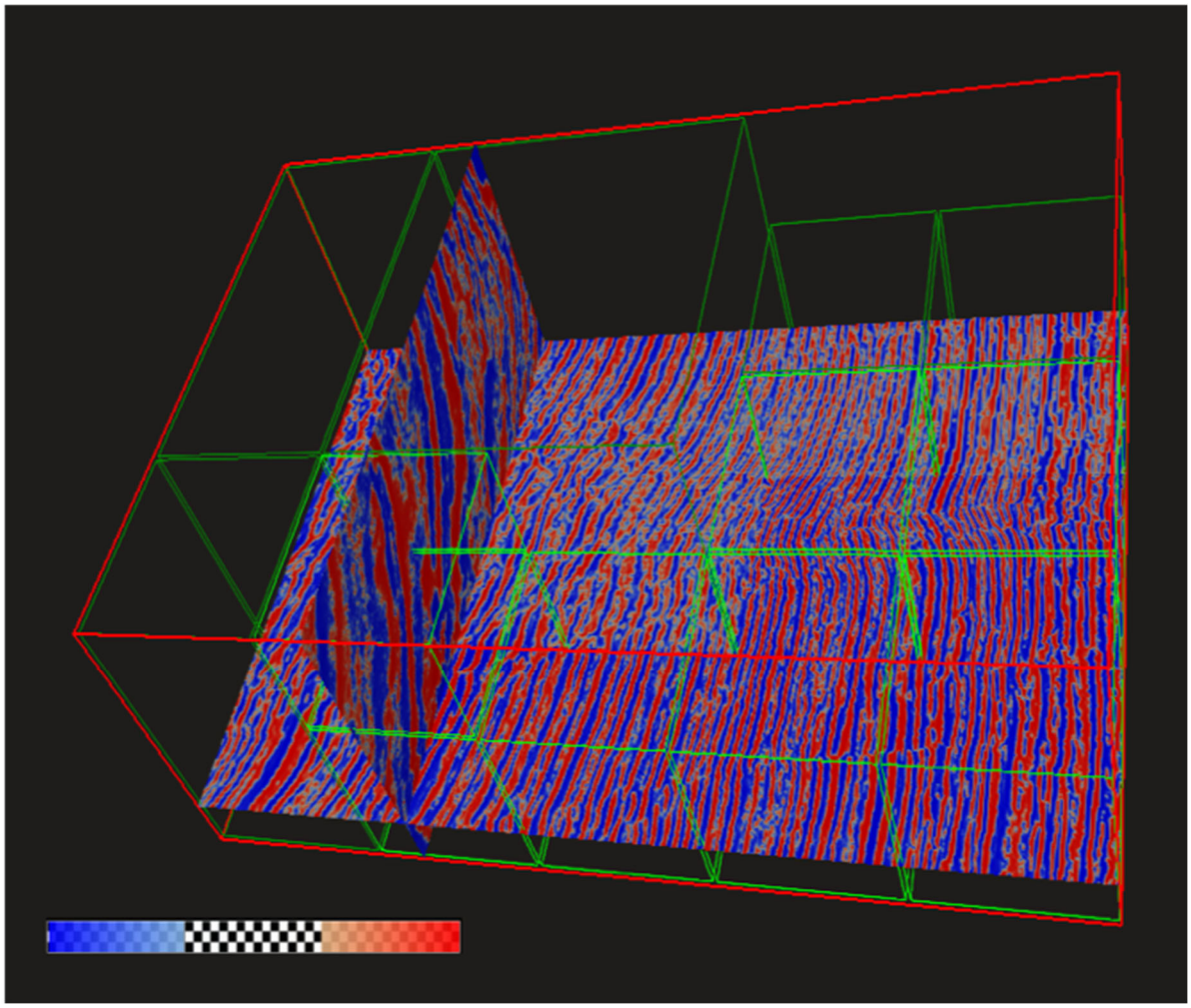


Figure 37: Slice rendering with a set of bricks from different resolution levels.

Implementation

Although significant improvements have been made in the Internet speed and quality, the cloud-based big data visualization remains a challenge to researchers since it is not feasible to transfer a few GBs from the cloud to any users at real-time. The data transferring latency requires us to implement a big data rendering service on the cloud so that only the rendered images are transferred between the users and the cloud, which make the real-time visualization feasible.

SAC implements a cloud-based big data analytics and visualization platform by providing seismic data accessing and visualization services. Figure 38 shows the user interaction with SAC through a web portal, the visualization module detects the users interaction with data, and then communicates with the data service and the rendering service. Data is sent from the data service to the render service to render them into 3D images, which are pushed to the web portal to display.

The first implementation of the cloud-based visualization takes over a minute to load a large seismic volume. The reason for the long-time visualization is that the 3D seismic volume is stored in memory aligned with one direction, either inline, crossline, or Z (or X, Y, Z). When the current display shows a seismic volume in 3D, whole volume is needed to render the image. To display a slice of data, SAC may need to transpose the data if the slice is not aligned with the internal data structure in memory, which takes significant time. To overcome these shortcomings, we implemented the brick format and the Level of Detail (LOD) to allow partial data transferring and data cache. Using LOD, SAC can cache different resolution of data on different levels. When users try to view the seismic data, the platform can show the low-resolution image to them at first. And then the higher-resolution data is gradually loaded into the memory in the background. The brick format removes the transposing task since a brick can be used for any directions of a slice.

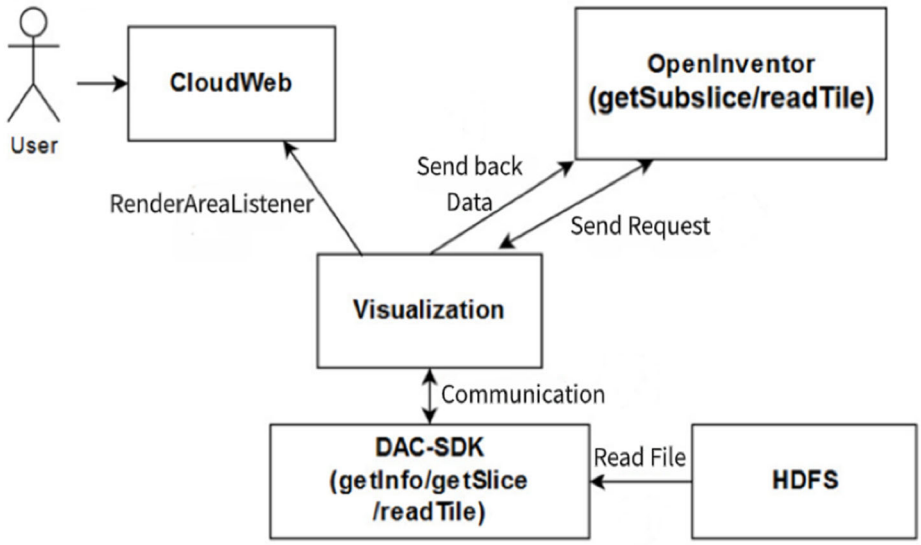


Figure 38: SAC data access and visualization services.

4.0 RESULTS AND DISCUSSION

4.1 Big Data Cloud Computing System

The PVAMU Cloud Computing lab has been working on building a scalable and shared cloud computing infrastructure to support research and education [268]. Fig 39 shows the Cloud Computing infrastructure. The infrastructure consists of three major components: 1) A Cloud center with a large number of Virtual Machines (VM) farm as the cloud computing service portal to all users; 2) A bare-metal high performance cluster to support HPC tasks and big data processing tasks; 3) a shared data storage and archive system to support data access and storage. In this system, the Cloud infrastructure functions as the service provider to meet a variety of users' requirements in their research and education. For HPC applications, the Cloud submits these tasks to the HPC cluster to fulfill their computing power demands. For those high throughput applications, the Cloud will deliver suitable VMs from the VM farm to meet their requirements. The Cloud orchestrates all functionalities of the entire system; provides elastic computing capability to effectively share the resources; delivers the infrastructure/platform services to meet user's research requirements; supports the big data storage and processing; and builds a bridge between end-users and the complicated modern computer architectures.

We are currently working on building a user-friendly, scalable and domain specific cloud [269] sponsored by NSF to deliver Platform as a Service (PaaS) to image processing researchers and developers. The cloud system is now able to store large amount of images and videos, as well as process them in parallel using Hadoop and Spark to achieve scalable performance. The goal is to enable image processing researchers and developers to write their algorithms using their favorite programming languages with very limited knowledge in parallelism, while be able to benefit from the scalable performance and large storage provided by the cloud. We have built a web-based programming interface to allow image processing researchers to manage their data, projects, and launch jobs on Hadoop with MapReduce and Spark computing engines. The MapReduce engine is used to support multiple languages programming environment to minimize user's revisions to their existing implementations. Spark is used to support high-level programming environment using Scala to allow users to take advantages of the abstraction, concise, and functional language

features. We have implemented spark RDD to support image formats to enable native data distributions and operations provided by Spark. Chapman's HPCTools research group, our collaborator from UH, is well known for their research in high performance and parallel computing, and programming and compiler support for accelerator and heterogeneous systems. For example, they have ported NASA Parallel Benchmark (NPB) suites [270] to GPGPUs architectures using high level OpenACC model [271]. The experience of creating the OpenMP and OpenACC benchmarks and testing suites [272, 273] will also be very helpful to users on how to integrate C/C++ based parallel programs with Spark/Hadoop systems. Chapman's group also develop in-house OpenUH [274] compiler and runtime system that support parallel computing and applications.

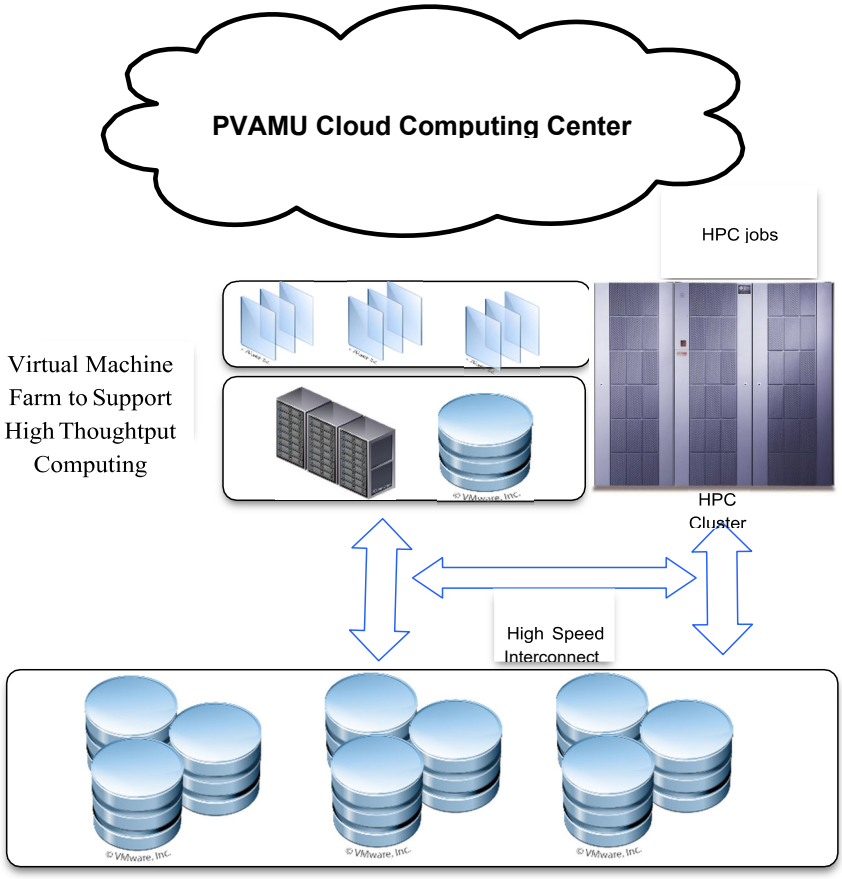


Figure 39: PVAMU Cloud and HPC Cluster for Big Data Processing

We will investigate the ideas of using OpenUH as online architecture-aware compilation and runtime adaptations framework to support upper-level Spark/Hadoop scheduling. The methodology and experiments of Chapman's previous work on parallel programming models [275{280], grid computing systems [281{283], scientific workflow scheduling systems [284,285] and the deployment of an air quality forecasting applications on computational grids [286, 287] will also be leveraged the proposed research work.

4.1.1 Domain-specific Cloud for Big Data Processing and Analytics

4.1.1.1 Experiments and Results

We have conducted numerous experiments on our 25 nodes of computer cluster located at Prairie View A&M University, in which one is master node and the other 24 are worker nodes. Each node of the cluster was configured with Intel Xeon E5-2640 Sandy Bridge CPU (2.5 GHz, 12 Cores), 64GB DDR3 memory. We have created a seismic data volume with 102GB, which is generated from the public Penobscot seismic data from OpendTect website with duplication and resampling. All of these experiments are performed with Spark 1.2.1 on Java 1.8.0 using different garbage collector setting [18] to be able to reduce garbage collection time as much as we can to improve the performance. Three test applications in seismic analysis are implemented and tested for the experiments: Seismic Calculator, Histogram, and Fast Fourier Transform (FFT).

We have run these applications using different numbers of CPUs to show the scalability. We also changed the data split granularity to test performance impact: using 1 inline, 10 inlines, and 30 inlines per split. All of these applications are tested in two ways: by running in Spark Shell using both cache option and un-cached one, and by submitting to Jobserver. We present the speedup by comparing with the corresponding sequential programs at the end. Spark performance web monitor UI, Spark Metrics and Nigels performance Monitor (nmon) are used to observe detailed information about running times and performance of these tests. Nmon Analyzer is used for following and observing cluster performance and finding the bottlenecks on the system.

4.1.1.2 SAC Web UI

Figure 40 shows the user interface of SAC. What user need for accessing seismic data hosted at cloud and verifying algorithm on it is only browser and an account. There are several tabs in SAC, such as Dashboard, Project, Datasets, Jobs, Workflow and some other useful tools. Dashboard will

give user a brief view about how many projects he/she had created and usage statistics of cluster. In Project tab, user could create new project, edit existing project, compile and run project. Jobs tab will show status of all running and finish jobs. User could view data sets and select on them to analyze in Datasets tab. Workflow is designed for complicate algorithms or batch jobs but still provide flexibility and usability to user for configuration.

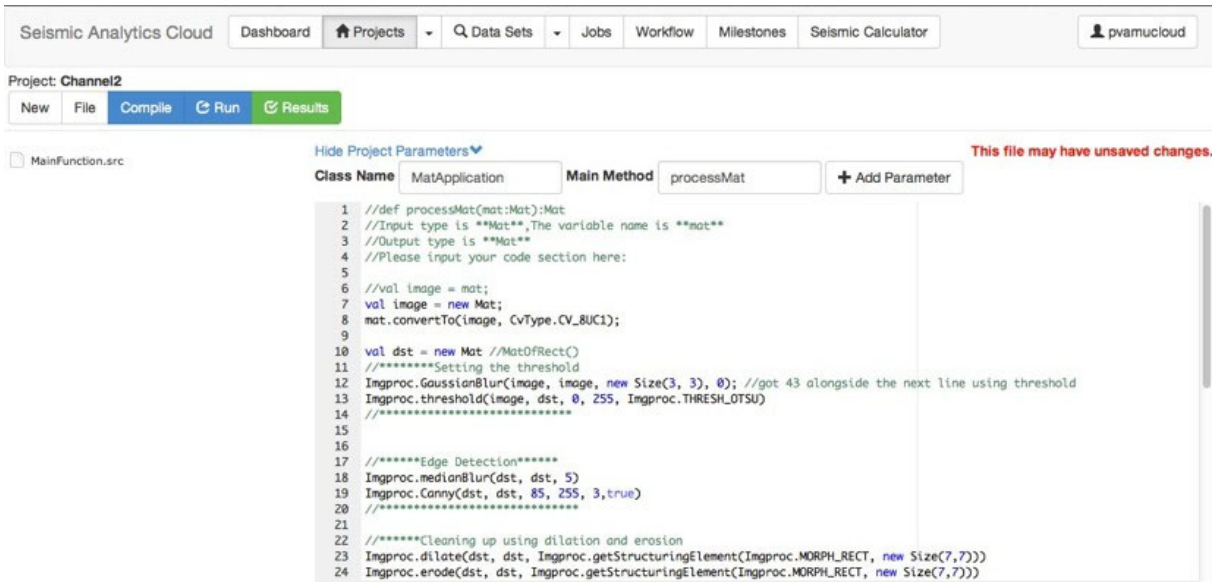


Figure 40: The SAC user interface

4.1.1.3 Seismic Calculator

Seismic calculation is a simple, useful but time consuming process when seismic data is big. In addition to the operations between two volumes, various types of arithmetic operations can be performed on a single seismic volume. These operations include arithmetic and logic ones that apply to every single sample in the volume.

4.1.1.4 Fast Fourier Transform (FFT)

FFT is the most popular algorithm for computing discrete Fourier transform (DFT), which is widely used in science and engineering. In seismic velocity model and image analysis, FFT is almost first and fundamental step. There are different implementations of FFT, such as FFTW, OpenCV, Kiss FFT, Breeze etc. Breeze is one of libraries in ScalaNLP, which includes a set of

libraries for machine learning and numerical computing. Spark itself already includes Breeze in its release, so we choose FFT algorithm in Breeze for experiment.

4.1.1.5 Histogram

This is the third application used for performance analysis. Histogram is to compute the data range distribution, which is used for estimation of the probability distribution of continuous quantitative variable. It is also a basic method for seismic data analytics. Spark already provides function to get histogram information from RDD directly. The bin size we choose for experiment is 10.

4.1.1.6 Performance Analysis

In this section, we will discuss the usability of SAC, and make deep performance analysis to find the bottleneck, which will also conduct performance tuning in the future.

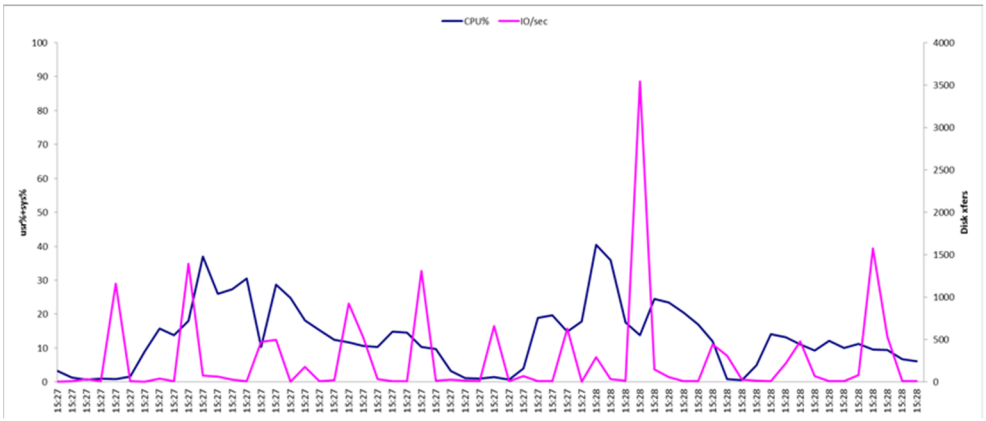
4.1.1.7 Usability Analysis

In the traditional seismic data processing methods using HPC, the product development flow requires a lot of geophysicists and IT developers involved: verifying algorithm with small sample data at first, then transferring into MPI codes with parallel optimization to handle actual big data. The whole process is time consuming and low efficient, and sometime even lead in consistent results between experiment data and actual data. On SAC, geophysicists and data scientists could verify their algorithms and directly experiment them with actual data. SAC could handle data distribution, code generation and execute the application in parallel automatically, but could provide fault tolerance natively and scalability. Take the 2D FFT case as example, user only needs to select template, write FFT algorithm or call other existing APIs, and type this piece of codes in SAC, in such function the input plane and output plane are already defined by SAC. The only things left are selecting data sets, compiling and running application, then viewing the results. In short, user only needs to take care about algorithm, and SAC will handle most of others, thus improve productivity apparently.

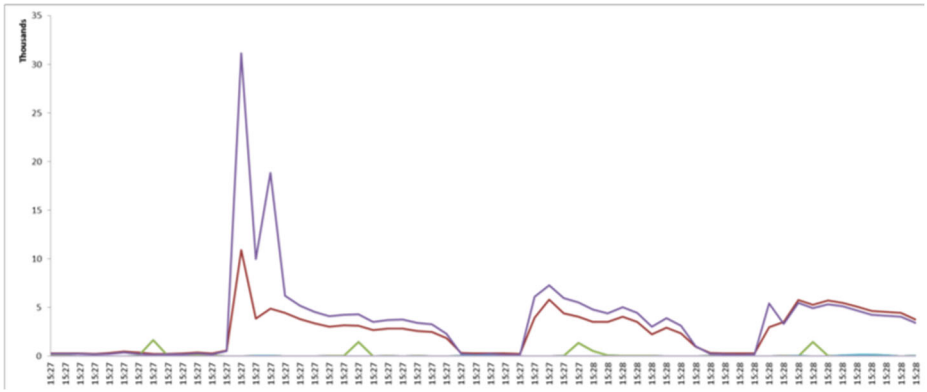
4.1.1.8 Performance Analysis of Seismic Calculator

Among all three different number of split sizes, the best results for calculator is achieved with 288 cores in first two, which indicates that more computing resource could get better performance. Closer look at the system with nmon-analyzer during run-time gives an interesting chart in network situation, CPU usage and the I/O of the system. Figure 41 shows these data versus each other.

Figure 41(a) shows CPU performance while on the other hand Figure 41(b) shows the network packets sending and receiving. It is obvious in the diagram that at the peak time for network CPU is not busy and at some points it became idle because of waiting for data. Increasing in network speed to have a better response for I/O request seems to be a key point in boosting the performance.



(a) CPU and I/O



(b) Network packets for calculator

Figure 41: CPU performance and network packets sending and receiving.

4.1.1.9 Performance Analysis of FFT

For FFT, it is a computing intensive workload hunger for CPU cycles instead of IO bandwidth. One system form the cluster was picked to show the performance characteristics in the run time. In Figure 42(a), CPU utilization quickly ramps up to 95% user time and mostly stays at the same level with several dips till the end of execution. There was not much time spend in kernel mode or

waiting for disk/network IO. There could be a little space for performance tuning to shorten the ramping up time in the start stage and remove the dips during the run. Figure 42(b) shows the disk read and write during the lifetime of the job. The maximum write is about 70 MB/s and the peak read is 50 MB/s. Both the read and write have not reached the bandwidth ceiling of the system. Same as the disk utilization, the network bandwidth was under 10 MB/s, which indicates underutilized network. The memory utilization in Figure 42(d), shows that memory was 60% occupied by FFT. The best results for this application are gained by using number of split size with 10 and number of cores 288. From the performance characteristics described earlier, FFT being a computing hunger workload, adding more computing power always will be beneficial, till other resources got over subscribed.

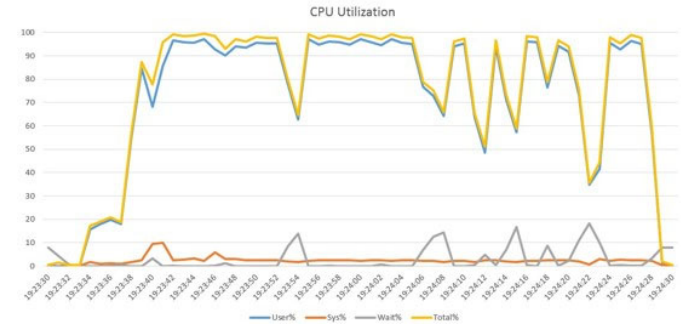
4.1.2 Implementing a Distributed Volumetric Data Analytics Toolkit on Apache Spark

To verify and demonstrate the DMATs scalability, we conducted a series of experiments, including data transposing and 3D stencil calculation, an overlapping-calculation application on our local big data cloud and the XSEDE supercomputing cluster.

4.1.2.1 Volume Transposing

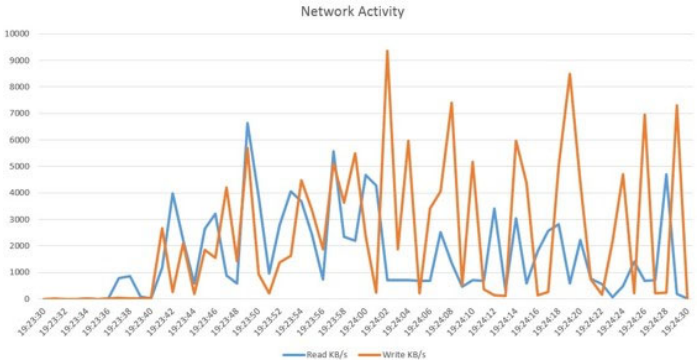
The dataset we used for transposing (from I to J direction) experiment is a 300GB seismic 3D volumetric data, which is 31017 x 97223 x 31 in I x J x K direction with float data type.

1. Scalability to the Number of CPU Cores: We conducted the transposing experiment on a cluster with 24 nodes, each node has 12 cores (24 cores in Hyperthreading) and 48GB available DRAM. The total CPU cores is 288(576 in Hyperthreading). Figure 43 shows the performance metrics of this experiment. From the metrics, we can see the performance scalability on dimension of the count of CPU cores is promised, as shown in line chart Figure 44 and Figure 45.

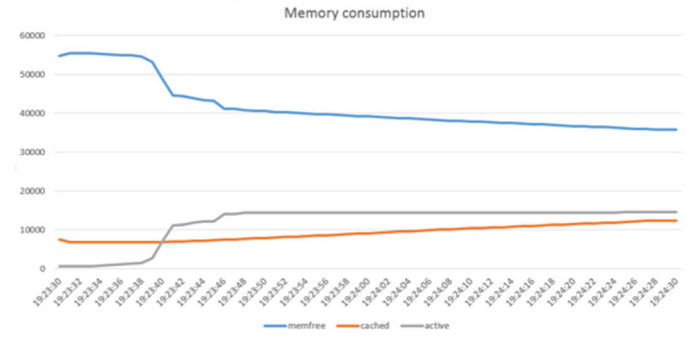


(a) CPU

(b) Disk I/O



(c) Network activity



(d) Memory utilization

Figure 42: Performance of FFT.

Cores	LoadFromFile(sec)	Transpose[1](sec)	Transpose[16](sec)	Transpose[64](sec)	Transpose[128](sec)
36	711.02	30246.483	604.083	444.867	433.9
72	531.849	15941.077	334.019	283.037	263.478
144	445.16	8865.005	210.677	178.212	168.96
288	404.325	5134.525	211.871	136.755	142.68
576	398.696	3914.722	179.026	131.292	141.675

Figure 43: The transposing experiment on Cluster with 288(576) cores

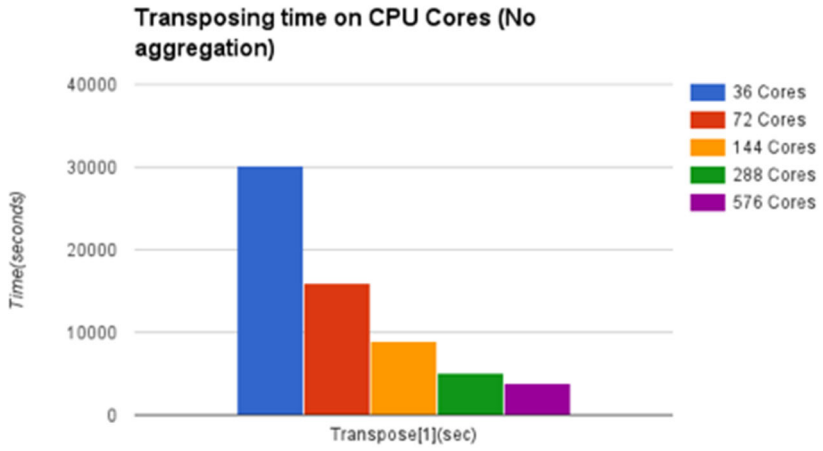


Figure 44: The transposing time on Cluster with 288(576) cores and 48GB memory per node

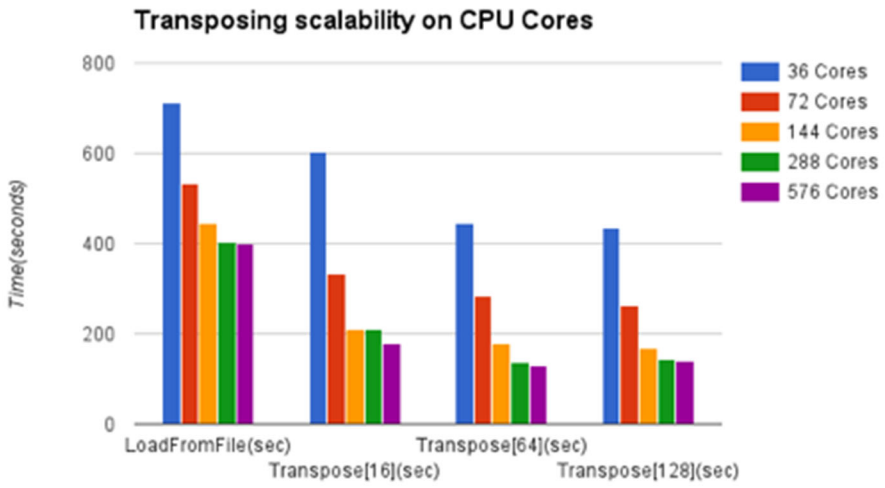


Figure 45: The transposing scalability on Cluster with 288(576) cores and 48GB memory per node

2. Scalability to the Fashion of Data Distributions: There is another important factor, the data distribution fashion, will also affect the performance of transposing. As we mentioned in previous section, the transposing program will perform some shuffle operation on volumetric RDD. Shuffle operations is Sparks mechanism for re-distributing data so that it's grouped differently across partitions. This typically involves copying data across executors and nodes, making the shuffle a complex and costly operation. To reduce the shuffle costs, we could aggregate the volumetric RDDs data splits to reduce the amount of data need to exchange during 3D transposing on each data split. Figure 46 shows the performance improvement when we increasing the aggregation planes count of each data distribution. However, the curve of performance trends to at when the aggregation increases to a higher level. That is caused by the performance trade-off when reducing the distribution partitions, which will lead to a lower cluster CPU cores utilizing rate. Therefore, to achieve a reasonable performance for transposing operation, developer needs to consider the trade-off between distribution configurations and data shuffling.

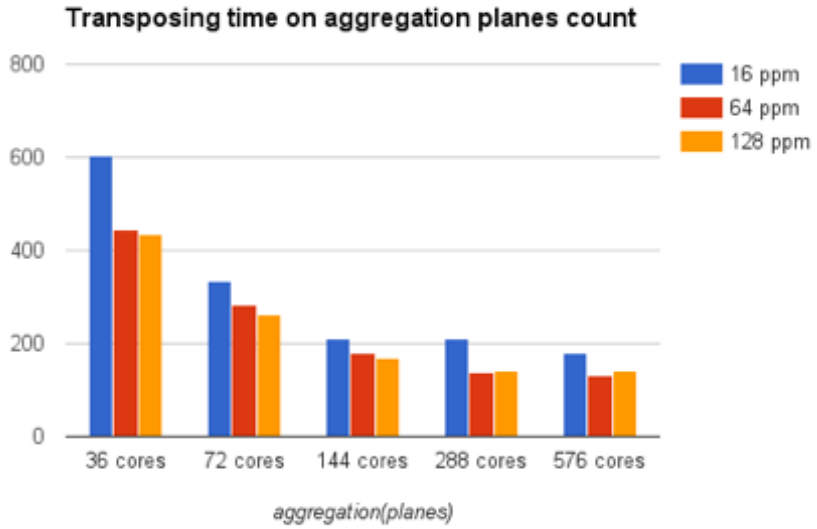


Figure 46: The performance on dimension of aggregation planes (ppm: planes- PerMap).

Figure 47 shows the CPU utilization and memory usage of each node when the transposing experiment was running on the cluster. These system statistic visualization views were generated by a free software NMONVisualizer, which is a Java GUI tool for analyzing Nmon performance files. To further verify the scalability, we also setup the same experiment on the XSEDE supercomputing cluster [11]. We request 44 nodes from XSEDE cluster, which has 12 cores in each node. As shown in Figure 48, we setup the experiment to test the scalability of case transposing the same volumetric data from I to J direction, aggregation is 16 planes, and the result also verified the scalability of this distributed application.

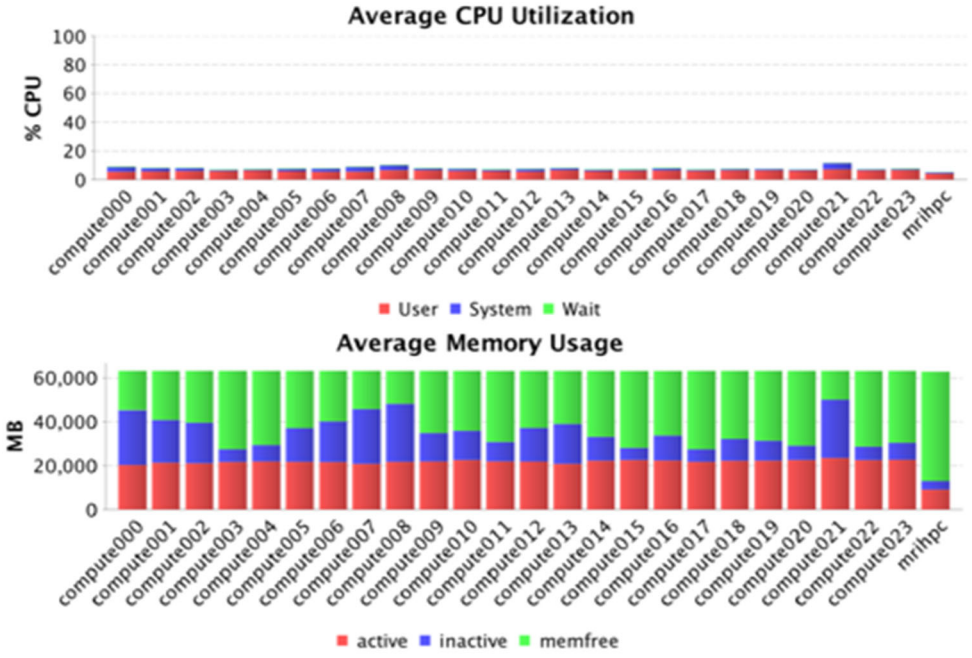


Figure 47: The runtime CPU and memory utilization statistics from NMONVisualier.

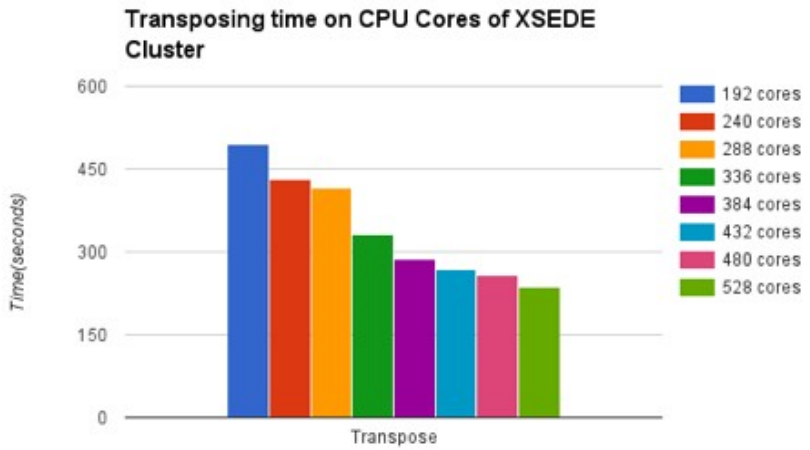


Figure 48: The transposing experiment on XSEDE Cluster.

4.1.2.2 3D Stencil Application

Stencil computations are most commonly used in context of scientific and engineering applications such as signal and image processing, computer simulations etc. Stencil itself represents an iterative kernel that updates array elements according to fixed pattens. The optimization on stencil computation has been well studied in [288, 289]. However, most of these optimizations focus on single node implementation with GPU or multi-core CPUs. In [267, 290], the authors provide a parallel implementation with Spark RDD, which gives a scalable solution for big data that cannot host on a single node.

In this work, we use the defined parallel templates in DMAT to implement stencil computations, and test their performance as well as scalability. The dataset we choose to conduct our experiments is called Penobscot dataset, which is actual seismic image data with 3D dimension size 600x481x1501. The cluster consists of 8 nodes, in which one is management node and other 7 nodes are computation nodes. Each node was equipped with Intel Xeon E5-2690 Sandy Bridge CPU (2.9 GHz, 16 Cores or 32 Cores with Hyper-threading support), 128GB DDR3 memory and are inter-connected with 1GB ethernet. JDK 1.8.0 40, Hadoop 2.5.1 and Spark 1.6.1 are used for compiling and running applications. Wall clock is used to get the running time, and Nmon/Spark

Web UI are used for performance analysis.

For the algorithm, we use a variant of Jacobi iteration, which uses a 3D subvolume with dimension size of 3x3x3 as input, and in the computation, each new output value at (i, j, k) is the average value of 26 surrounding samples plus itself. In the case of 3x3x3 subvolume, the overlap area is 1. For the sequential codes, we just split the big 3D data file into small partition and each partition includes several 2D planes (the overlap between partition is one 2D plane), and then use 3 nested loops to compute the average value. For the parallel codes using DMAT, we use the overlapped template, which specify parameters both in I and J directions. Different configurations of cores and numbers of planes in each partition are set to check performance and scalability. The numbers of cores (28, 56, 112, 224) are used for each test case respectively to verify the scalability of parallel codes. Within each configuration of cores, we use different combinations of dimension size (1, 2, 4, 8) in I and J directions. For instance, I(4) and J(2) mean that the dimension size of input subvolume 6x4, which comes from (4+2*1)x(2+2*1) in the case of overlap is 1, which is shown in Figure 4.11.

Figure 50 and Figure 51 show the speed up of parallel codes on Spark with 28 cores and 224 cores to the sequential codes, respectively. From these two figures, the changes on number of J have little impact on the performance, because it does not change the number of planes in each partition and the number of partitions. In the template implementation, two nested loops are used for feeding the input of each stencil kernel. However, the change on number of I tells how the SDK distributes the data and the number of partitions, thus will determine how many tasks are need to finish that stage of in the job, and each task need to be assigned one thread or one core to undertake the computation. In the case that size of I is 1, it gets the best speed up in all test cases. It seems to be abnormal that the performance decreases from 1 plane of I to 2 planes of I. Increasing I will enlarge the size of each partition as shown in Figure 49, however, the amount of computations keeps constant. In this stencil computation, we iterate several times to reach the balance. At the beginning of each loop, the data need to be repartitioned based on the overlap parameter, since the latest edge data need to be updated to each partition for next iteration. In the process of repartition, it needs to get planes at the left and right edge, sort them by key and zip with original latest results, which trigger data shuffle in Spark. The bigger the size of partitions is, the more time it takes to shuffle them. The Shuffle Read Blocked Time increases drastically from 1 plane of I to 2 planes of I, which accounts for the performance decreasing. Figure 52 shows the best speedup with different configurations of cores, in which the scalability is obvious while increasing the number of cores.

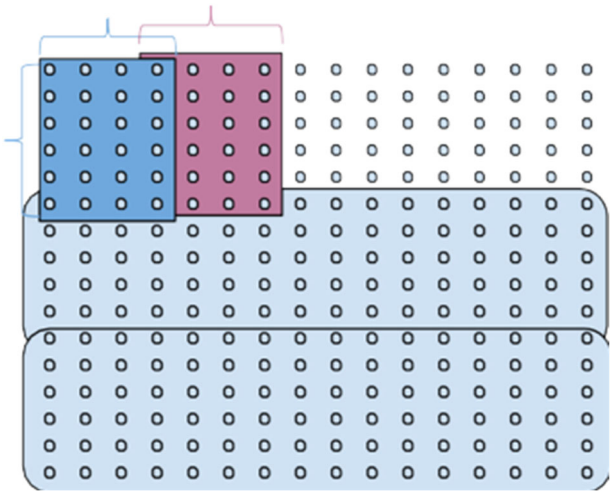


Figure 49: The Data Distribution and Input of Overlap Template.

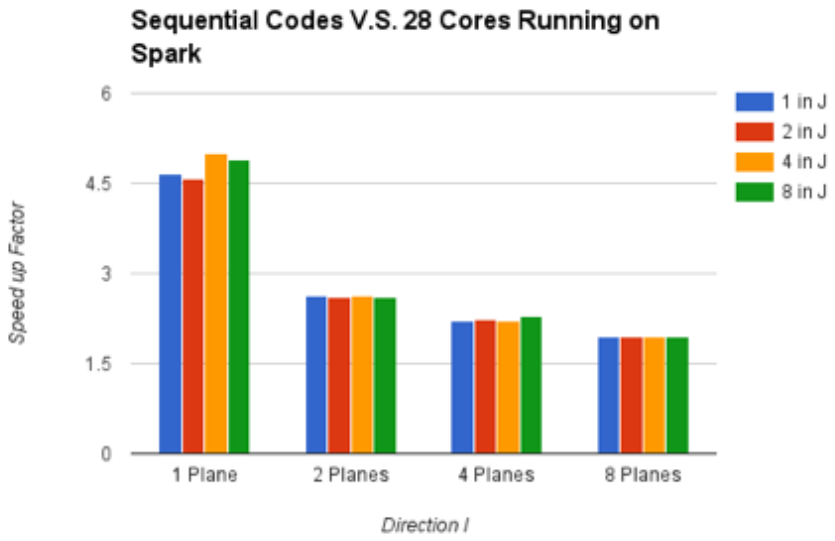


Figure 50: The Speedup of Parallel Template Codes with 28 Cores to Sequential Codes.

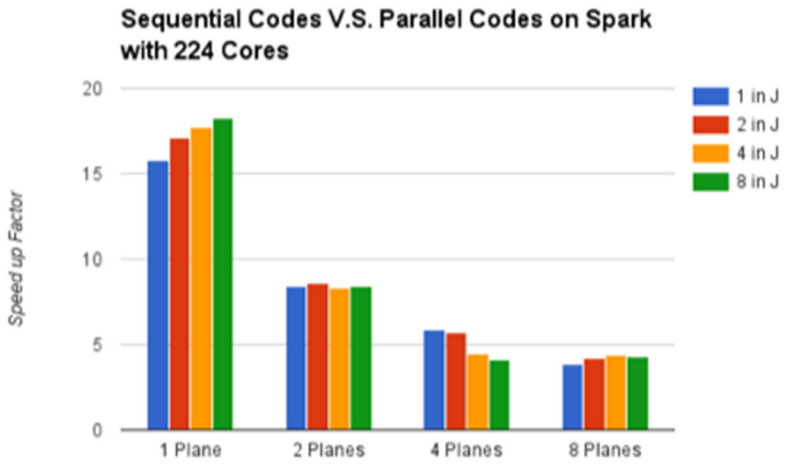


Figure 51: The Speedup of Parallel Template Codes with 224 Cores to Sequential Codes.

.

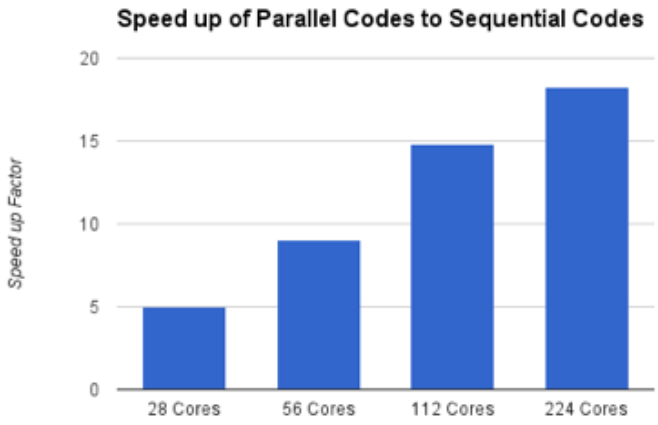


Figure 52: The Best Speedup of Parallel Templates for Stencil Computation.

4.2 Reliable and Robust Data Collection and Aggregation

4.2.1 Efficient privacy preserving edge computing for images and video

4.2.1.1 Dataset

Dataset Description:

The result in this work is generated using three different datasets summarized in Table 2. These datasets are from the Canadian Institute For Advanced Research dataset (CIFAR10) [291] and the ILSVRC (ImageNet) 2012 datasets [292].

Canadian Institute For Advanced Research (CIFAR10):

This dataset containing 60,000 color images is a subset of about 80 million labeled but tiny images. The dataset is further divided into 50,000 training samples and 10,000 testing samples, each of dimension $32 \times 32 \times 3$. It has ten (10) mutually exclusive classes with no semantic overlaps between images from different classes.

ILSVRC (ImageNet) 2012:

The original ILSVRC 2012 dataset contains about 1.2 million color images of different sizes across about 1,000 classes. The 1,000 classes are either internal or leaf nodes but they do not overlap. Two subsets of the ILSVRC 2012 dataset termed IMGNET-A and IMGNET-B are used in this

work. Each subset contains about 13,000 images each resized to a dimension $224 \times 224 \times 3$, spanning 10 classes. Each subset dataset is further divided into training samples and testing samples with a ratio of 7:3. The difference between the two subsets lies in the type of nodes they contain. The IMGNET-A subset contains images from 10 different leaf nodes (diverse images), while IMGNET-B contains ten (10) child nodes from a single leaf node (similar images).

4.2.1.2 Deep Learning Model Design and Training Strategy

The autoencoder for the edge devices and the classifier at the edge server are chosen because the autoencoder is optimized for feature extraction and the classifier is optimized for image classification.

The autoencoder architecture is affected by the type of images and the compression ratio. For instance, the model architecture for the CIFAR10 dataset for compression ratios 4 and 8 are different. This condition also applies to compression ratio 4 for IMGNET-A and CIFAR10 datasets. Hence, different models are developed across several edge devices, compression ratio, and datasets.

Figure 53 shows the model architecture for an encoder designed for the CIFAR10 dataset for a compression ratio of 4. In general, the autoencoder model contains a mix of convolutional (same padding), max pooling, and upsampling layers. The ReLu function is used as the activation function for all layers except the last layer, where the sigmoid function is used.

In this work, the autoencoder models are trained from scratch using the glorotuniform method as the initializer, mean square error as the cost function, and rmsprop optimization algorithm as the optimizer. After the convergence of the autoencoder model during the training process, the encoder part of the autoencoder is then extracted, and deployed in the inference mode to compress all the images to obtain the latent variables needed to train the classifier. The mean square error (MSE), which also doubles as the cost function is used as the metrics of the autoencoder.

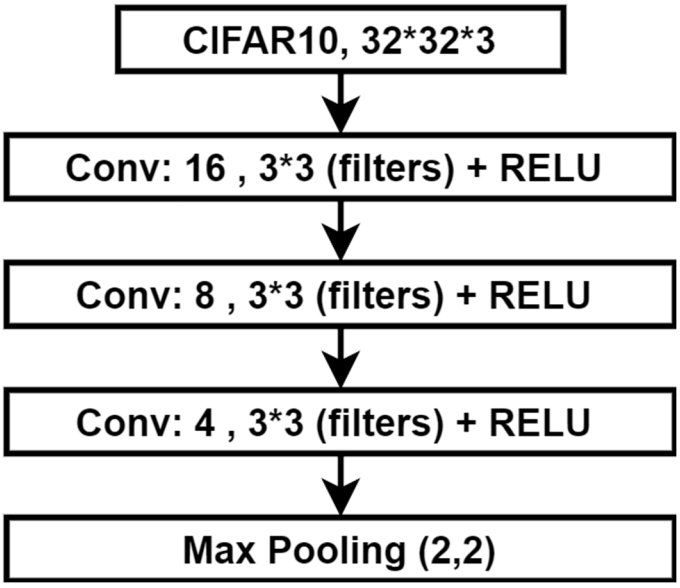


Figure 53: Details of an encoder model for compression size of 4 using CIFAR10 dataset

Training Stage: Classifier Design

The convolutional neural network (CNN) model is used as the classifier in this work. CNNs are well suited for image processing applications and other grid-like data [127]. They are more computationally efficient than the dense deep neural network (DNN), thus reducing memory usage. Using the filters, CNNs find and extract meaningful features from the images and preserve spatial relations. Three different CNN classifiers, denoted Model-A, Model-B, and Model-C, as listed in Table 3, are used in this work.

Model-A and Model-B:

Model-A and Model-B are considered to be vanilla models because they are trained from scratch. Model-A and Model-B are specifically designed for the original input image and feature maps of the CIFAR10 dataset and ImageNet dataset, respectively. The detailed CNN architecture of Model-A and Model-B are shown in Tables 7 and 6, respectively. The models contain a mix of convolutional, max pooling, and fully connected layers. The ReLu and softmax activation functions are also used for the model design.

Furthermore, the models also contain some dropout layer in order to prevent over- fitting. The differences between Model-A and Model-B lie in the number of the various layers used and padding of the convolutional layers of Model-A.

The models are trained from scratch to minimize the difference between the labels (ground truth) and the predicted labels. This training is achieved by the use of the glorot-uniform method as the initializer, categorical cross-entropy as the loss function, and Adams optimization algorithm as the optimizer. Data augmentation is also used during the training process to mitigate overfitting due to the small quantity of the datasets. It should be stated that each classifier is trained with their respective original image and the feature maps (compressed images).

Model-C:

Model-C is a transfer learning based model explicitly designed for the ImageNetdataset in this work. The CIFAR10 version of the result is not presented in this work as the compressed data gives a poor performance with the transfer learning models. This poor performance can be attributed to the small dimensions of the CIFAR10 dataset and large depth of the various transfer learning models used.

The block diagram of the model is shown in Figure 54. Model-C can be divided into two parts: The base layer and the top layer. The base layer is a pre-trained layer of another standard deep learning model (without the fully connected layer) trained with data similar to the ImageNet data and achieved better performance. Using this pretrained model, the excellent feature extracting property of the standard model is being leveraged to achieve better performance. Furthermore, it also complements data augmentation in training a decent model in situations where datasets are limited. VGG16, VGG19, InceptionV3, InceptionResnetV2 and Resnet50 pre-trained models [293] are used as base models for Model-C.

The details of the top layer used for this work are shown in Table 8. It should be noted that the first dense layer of the top layer in the fifth model (Model 5) is smaller than that of the other models. Model 5 suffers from overfitting if the number of neurons in the first layer is 256, the same number used in the other models. Hence, the size of the dense layer is lowered to reduce overfitting and achieve good performance.

Table 6: The architecture of the vanilla model for CIFAR10 dataset (Model-A)

Vanilla Model For CIFAR10 Dataset
1*Conv2D, Filter Size=3*3, No of Filters=32, Stride=1*1,Padding
1*Activation Layer (Relu)
1*Conv2D, Filter Size=3*3, No of Filters=32, Stride=1*1, Padding
1*Activation Layer (Relu)
1*Max Pooling, Pool Size = 2*2,Stride = 1*1, Padding
1*Dropout(0.25)
1*Conv2D, Filter Size = 3*3, No of Filters = 64,Stride = 1*1, Padding
1*Activation Layer (Relu)
1*Conv2D, Filter Size = 3*3, No of Filters = 64,Stride = 1*1, Padding
1*Activation Layer (Relu)
1*Max Pooling,Pool Size = 2*2,Stride = 1*1, Padding
1*Dropout(0.25)
1*Flatten
1*Dense(512)
1*Activation(Relu)
1*Dropout(0.5)
1*Dense(10)
1*Activation(Softmax)

Table 7: The architecture of the vanilla model for ImageNet dataset (Model-B)

Vanilla Model For ImageNet Dataset
1*Conv2D, Filter Size = 3*3, No of Filters = 32, Stride = 1*1,No Padding
1*Activation Layer (Relu)
1*Max Pooling, Pool Size = 2*2, Stride = 1,1,No Padding
1*Conv2D, Filter Size = 3*3, No of Filters = 32, No Padding
1*Activation Layer (Relu)
1*Max Pooling, Pool Size = 2*2,Stride = 1*1,No Padding
1*Conv2D, Filter Size = 3*3, No of Filters = 32,Stride = 1*1, No Padding
1*Activation Layer (Relu)
1*Max Pooling,Pool Size = 2*2,Stride = 1*1,No Padding
1*Flatten
1*Dense(64)
1*Activation(Relu)
1*Dropout(0.5)
1*Dense(10)
1*Activation(Softmax)

A 2-stage training method is used for the transfer learning model to minimize the error between the ground truth labels and the predicted labels. This approach is different from the training approach used for Model-A and Model-B, which are trained from scratch. In the first stage, the base layer is fixed while the fully connected top layer is trained using the Adam optimizer after being initialized using the glorot-uniform method. This approach is taken to initialize the weight of the top layer close to the weight of the base layer. The complete model is then retrained, and all the weights are appropriately tuned using the stochastic gradient descent (SGD) with momentum optimizer. SGD with momentum is used because it is less aggressive than the Adam optimizer. The use of an aggressive optimizer in the second step might cause the weights (information) in the base layer to be significantly eroded or lost. The categorical cross-entropy is used as the cost function in the entire training process.

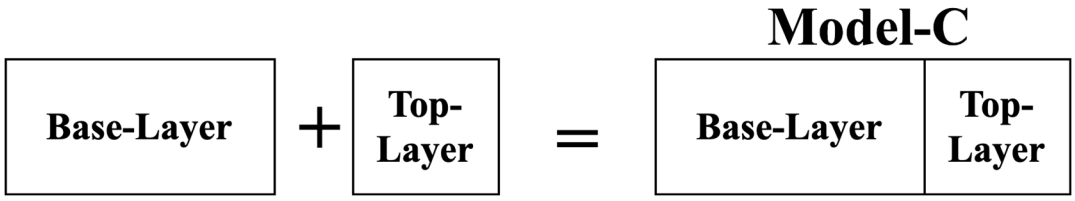


Figure 54: The Transfer Learning Model Block (Model-C)

Table 8: The architecture of the transfer learning model for ImageNet datasets (Model-C)

	Model 1	Model 2	Model 3	Model 4	Model 5
Base layer	VGG16	VGG19	InceptionV3	InceptionResnetV2	Resnet50
5*Top layer	Dense(256)	Dense(256)	Dense(256)	Dense(256)	Dense(50)
	Activation(Relu)	Activation(Relu)	Activation(Relu)	Activation(Relu)	Activation(Relu)
	Dense(10)	Dense(10)	Dense(10)	Dense(10)	Dense(10)
	Activation(Softmax)	Activation(Softmax)	Activation(Softmax)	Activation(Softmax)	Activation(Softmax)

4.2.1.3 Experiments

This work seeks to propose a new approach to design and implement deep learning models for distributed systems without compromising data privacy and security. It achieves this by extracting the most important/critical machine intelligible features but human unintelligible features from the dataset. These features are then transmitted across the communication network from the edge devices to the edge server, where they are aggregated and used to train a classifier. The experimental

methods, performance metrics, and tools used in validating our proposed framework are explained in this section.

A 2-stage methodology is used to validate our proposed framework, and this method is the same irrespective of the type of dataset or model used. In the first stage, the training set of the original input dataset (uncompressed images) is used to train the classifier. After that, the test set is used to obtain the needed performance metric to set the baseline performance.

In the second stage, the training set of the feature maps (compressed images of the dataset used in stage 1) is used to train the same classifier model. The feature map, which is smaller than the original image by a pre-determined factor, is obtained by passing the original dataset through the autoencoder's encoder. After that, the performance metric of the classifier is obtained using the test set of the feature maps and the performance compared to the baseline performance.

Performance Metric:

The effectiveness of the framework is assessed using a simple classification task. The test accuracy of the model obtained after the training process is used as the primary performance metric although the F-score measurement of the models is also obtained to further validate the performance of the models. Furthermore, our proposed framework's effect on the training time, testing time, and the number of model parameters is also investigated. It should be noted that the primary performance metric for this section of the framework is application specific. For Natural language processing applications for example, this metric will change to either of Cosine Similarity, Jaccard Similarity, Perplexity or Word Error Rate.

Software and Hardware:

The design, training, and testing of the deep learning models (Autoencoders and CNN Classifiers) are implemented using Keras deep learning framework on TensorFlow backend, running on an NVIDIA Tesla P100-PCIE-16GB GPU.

4.2.1.4 Results and Analysis

The results of the experimental work are presented in this section. The proposed framework's performance is compared with our baseline using the performance metrics stated above. The baseline performance is represented by compression ratio one (1), and it is synonymous with using the

uncompressed image to test our various models. Furthermore, it should be noted that the vanilla model for the CIFAR10 and ImageNet datasets are different.

Figure 55 shows the testing accuracy of vanilla CNN Classifiers (Model-A and Model-B) when trained and tested with compressed and uncompressed CIFAR10 and ImageNet datasets. The testing accuracy for the compression ratio 1 (uncompressed images), representing our baseline, is highest across all the cases, as expected. This observation is because all the features in the raw images are used for the classification task. Furthermore, the testing accuracy for IMGNET-A is larger than the testing accuracy of IMGNET-B. The differences in performance can be attributed to the very close similarity in the images in IMGNET-B, as classifying such images is a much more difficult classification task than classifying images in IMGNET-A. The classifier requires more information than what is available to identify each of the class in IMGNET-B than IMGNET-A uniquely.

A general degradation in the testing accuracy is observed in Figure 55 as the compression ratio is increased, although the rate of degradation varies across the models used for the three (3) datasets. The rate of degradation of the testing accuracy of the model trained with CIFAR10 dataset is the highest for all the compression ratios. The observed degradation is because the small dimension of the CIFAR10 images ($32 \times 32 \times 3$) implies that the number of features needed to perform a classification task is even smaller when compressed. This means the information contained in the image has been reduced, making it difficult for the model to have enough information to identify each class. Furthermore, the rate of degradation of the testing accuracy for the IMGNET-A dataset is very modest across all the compression ratios. However, similar performance is not observed in IMGNET-B, particularly for compression ratios 8 and 16 despite having the same image dimension ($224 \times 224 \times 3$) as the IMGNET-A dataset. The larger degree of degradation observed in the model trained on IMGNET-B for compression ratios 8 and 16 is due to the complexity of the classification task. The degradation is because of the similarities in the images that make up the various classes in IMGNET-B, which indicates more features are needed to identify the image for each class uniquely. The considerable diversity in the IMGNET-A dataset classes makes the classification task less complex and requires fewer features to identify each class and differentiate between the classes uniquely. This reduced complexity explains why it suffers low degradation in testing accuracy even at a higher compression ratio despite fewer features being used for classification. In order to further validate the performance of the models, the F-score of the models is

obtained and shown in Figure 56. It can be observed that the F-score values of the models for various compression ratios are approximately the same as the corresponding model accuracy values. Furthermore, the F-score values show a similar trend as the testing accuracy when the compression ratio is increased. This is expected because the dataset used for this work is a balanced dataset. The testing accuracy of the transfer learning based model (Model-C) for the ImageNet dataset compressed by a factor of 4, using different base models, is shown in Figure 57. The transfer learning model is not designed and trained using the CIFAR10 datasets as its performance is poor with the compressed images. The poor performance can be attributed to the deep nature of the transfer learning based model. Due to its depth, the transfer learning model has an inadequate number of features available at the fully connected layer of the model (top layer) where classification takes place. Hence, there are inadequate features available for the classes in the dataset to be uniquely identified. The same reason also explains why the transfer learning model is only designed and tested with the ImageNet dataset with a compression ratio of four(4). At higher compression ratios (8 and 16), the performance is poor with the compressed images as there are fewer features at the fully connected layer of the model (top layer) for the model to uniquely identity each class in the IMGNET-A and IMGNET-B dataset.

From Figure 55 and Figure 57, it is observed that the testing accuracy of the transfer learning model across different base models for IMGNET-A and IMGNETB datasets at compression ratio 1 (baseline) and 4 is higher than the corresponding performance of the vanilla model (Model-A and Model-B). This phenomenon can be attributed to the powerful feature extraction property of the various base layers used in the transfer learning model. The base layer is a neural network of various configurations or architectures trained on the complete ImageNet dataset for the classification task. The base layer has powerful feature extraction property because it has been trained on a classification task similar to the classification task at hand and achieves satisfactory testing accuracy.

As observed with the vanilla model in Figure 55, the testing accuracy for compression ratio 1 (baseline) is better than the testing accuracy for compression ratio 4 for both IMGNET-A and IMGNET-B datasets with the transfer learning model as well (Figure 57). Furthermore, the performance of the transfer learning model trained on the IMGNET-A dataset is better than that of the model trained on the IMGNET-B dataset for compression ratios one (1) and four (4). However,

the rate of degradation in the testing accuracy for compression ratio 4 is higher than what is observed for the vanilla models (Figure 55) for both IMGNET-A and IMGNET-B datasets. The degradation can also be attributed to the deep nature of the transfer learning models as the small amount of information/features available at the fully connected layer of the model (top layer) are not distinct enough to make an accurate classification. The number of parameters in a convolutional neural network is determined by many factors such as the filter size, the number of filters, the size of the input data, the number and type of hidden layers. Hence, a reduction in the number of trainable parameters can be achieved by reducing the size of the input data. The relationship between the normalized number of parameters in the vanilla model for the CIFAR10 and ImageNet datasets vs. the compression ratio is shown in Figure 58. It can be observed that for the same compression ratio, the rate of reduction in the normalized number of parameters of the vanilla model for the ImageNet dataset is much bigger than that for the CIFAR10 dataset.

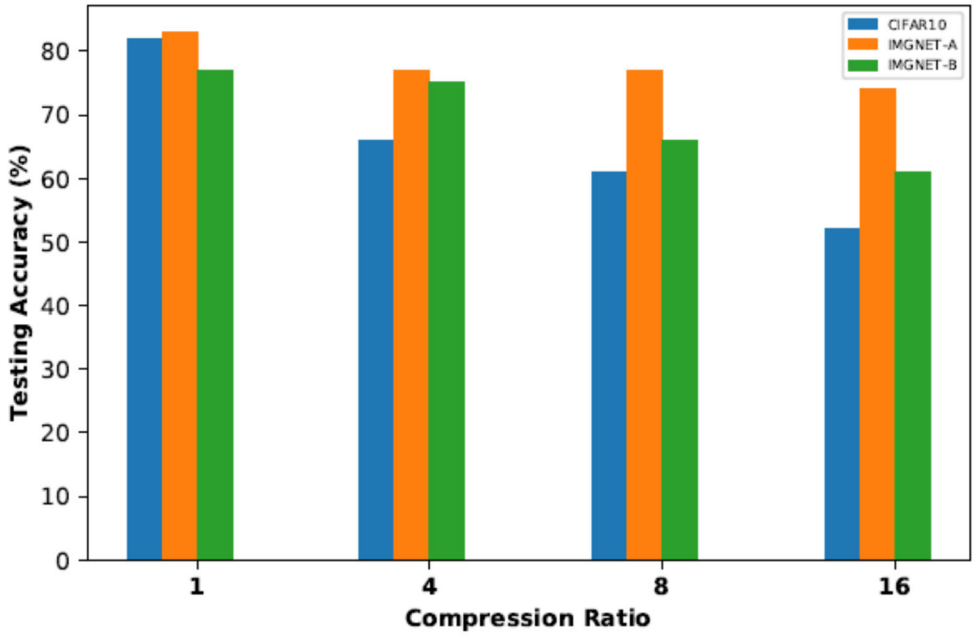


Figure 55: Comparison of the testing accuracy of the vanilla models for the original dataset (compression ratio =1) and compressed dataset (latent variables) with compression ratio = 4, 8, 16.

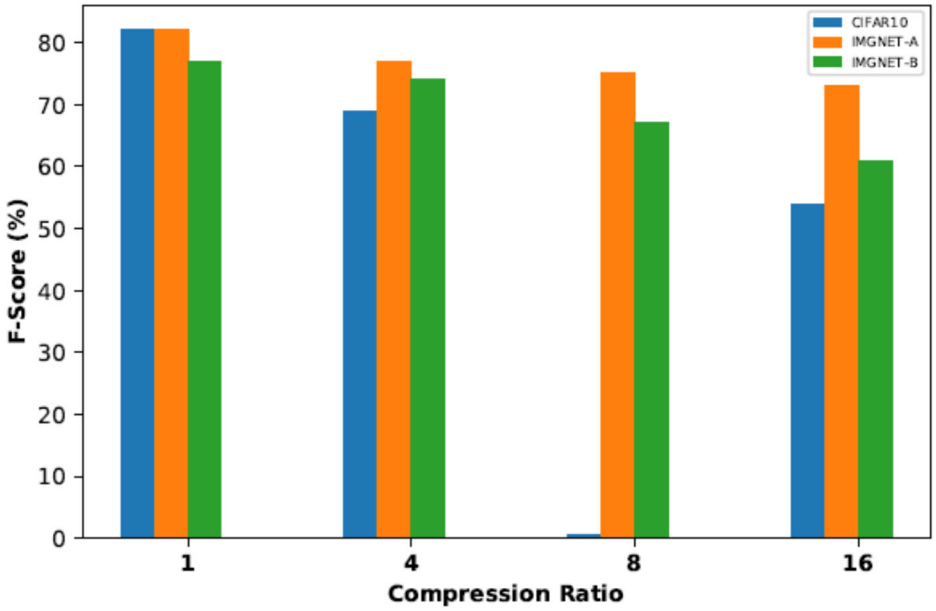


Figure 56: Comparison of F-Score of the vanilla models for the original dataset (compression ratio =1) and compressed dataset (latent variables) with compression ratio = 4, 8, 16.

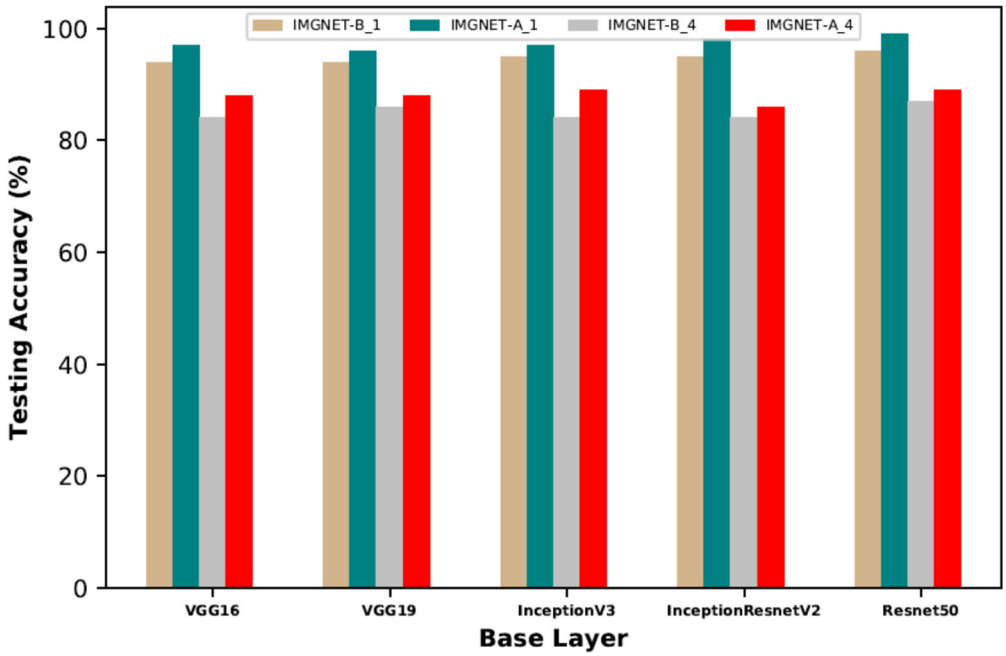


Figure 57: Testing accuracy of the transfer learning based model (Model-C) using different base models for the ImageNet dataset with compression ratio = 4.

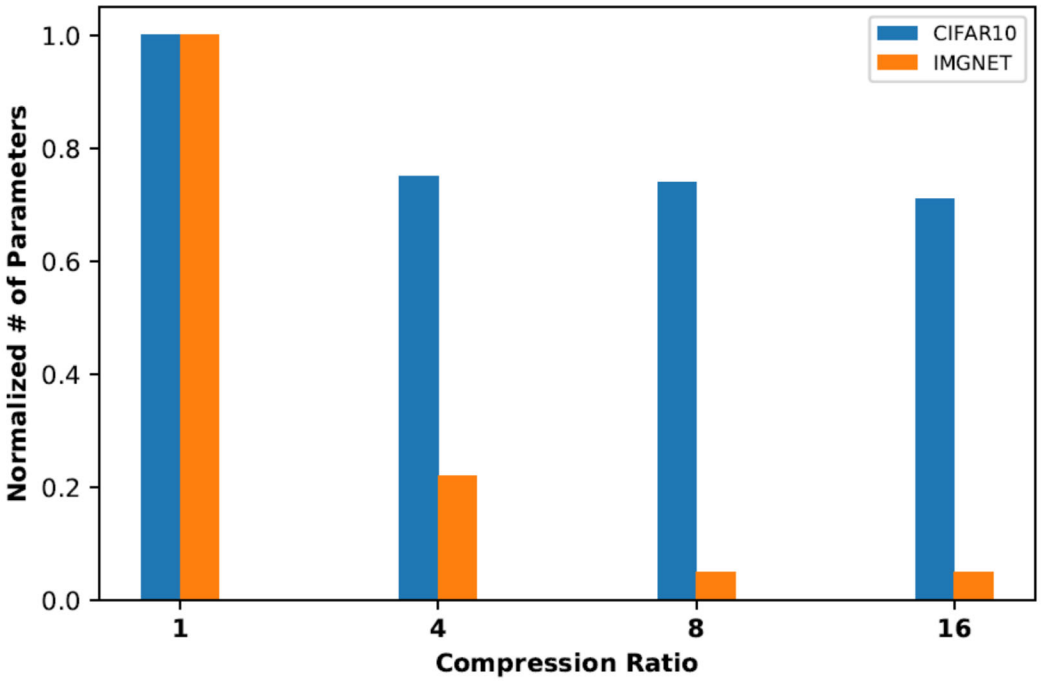


Figure 58: Comparison of the normalized number of vanilla model parameters vs. data compression ratio

This is because when the compression ratio increases, the already low-resolution images from CIFAR10 cannot be reduced much further by the model, and the number of parameters needed will remain almost constant for compression ratios 4, 8, and 16. On the contrary, the number of required parameters will keep decreasing in the case of the ImageNet dataset for compression ratio 4 and 8 because the images have much higher resolution, and as a result, the parameters will keep decreasing when the input image becomes smaller. However, when the compression ratio is 16 for ImageNet, the images are already small, they cannot be reduced much further, and thus will not affect the number of parameters.

Figure 59 shows the normalized amount of time required for testing and training of the vanilla models at various compression ratios for CIFAR10 and ImageNet datasets, respectively. A reduction in training and testing time is observed across the compression ratios and the models. The reduction can be attributed to the decreasing size of the input data, which directly affects the total number of trainable parameters. As the size of the input images decrease with the increase in compression factor, the total number of trainable parameters at the fully connected layer and the total number of trainable parameters also decrease. The total number of trainable parameters in a model

indicate the degree of complexity of the resulting model and the amount of time required to train the model (training time) and test the model (testing time). Hence, the reduction in the training time and testing time as the compression factor increases.

Although Figure 59 and Figure 58 represent a desirable improvement in some of the properties of the resulting model, such as training time, testing time, and the number of parameters as the compression factor increases, the model accuracy, as shown in Figure 57 reduces with increase in the compression factor. These Figures show the trade-off between the degree of privacy desired and the model accuracy. It also represents a trade-off between our proposed method and the compression factor of one, which is equivalent to server-only computation. Our proposed framework leverages on both the resources at the edge and at the cloud server. The edge computation part (autoencoder) helps extract the useful features needed for training at the cloud server. The cloud server provides the resource needed to train the deep learning model (classifier). Training only at the edge will lead to poor performance as there are no sufficient data and edge device cannot handle large deep model due to limited computing resource. Training only at the server will require all data available at the server, which might not be possible due to privacy concerns and limited bandwidth. Hence, there is a need to find an optimal trade-off point between the compression and model accuracy.

Although the testing accuracy is used as the primary metric for the framework in this work, this could change as the choice of this metric is application specific. The choice of testing accuracy as the primary metric in this paper is informed by the use of an image classification model as the second model. With object detection application, average precision or intersection over union metric can be used to judge the effectiveness of the model. In cases where a metric that is application dependent is desired, the use of the mean squared error (MSE), which is the cost function used in training the autoencoder can be considered. The MSE is the difference between the encoder input and the decoder output. This metric is the same irrespective of the type of application the framework is used for. The lower the MSE, the more the effectiveness of the encoder model in extracting the latent variable/features in the input. Furthermore, there is a strong negative correlation between the MSE and the testing accuracy. This is because the output of the pre-trained encoder is used in training the classification model in the second part of the model. Hence, the lower the MSE of the encoder, the better the accuracy of the resulting model trained with the output of the encoder. The plot of the MSE of the encoder used in generating the compressed input for Model-A and Model-

B is shown in Figure 60. It is noticed from the table that the MSE increases with an increase in the compression ratio for a particular dataset, which speaks to the inverse relationship between the MSE of the encoder and the testing accuracy of the classification model. It is also noticed that the MSE value of the encoders for ImageNet-A dataset is bigger than the MSE value of the encoders of ImageNet-B dataset. This same trend is noticed in Figure 55 where the testing accuracy of the models trained with compressed images of ImageNet-A is bigger than those of ImageNet-B. This behavior is attributed to the degree of complexity of the images in ImageNet-B dataset due to their close similarity as compared to images in ImageNet-A.

4.2.2 Computation offloading

4.2.2.1 Performance Analysis

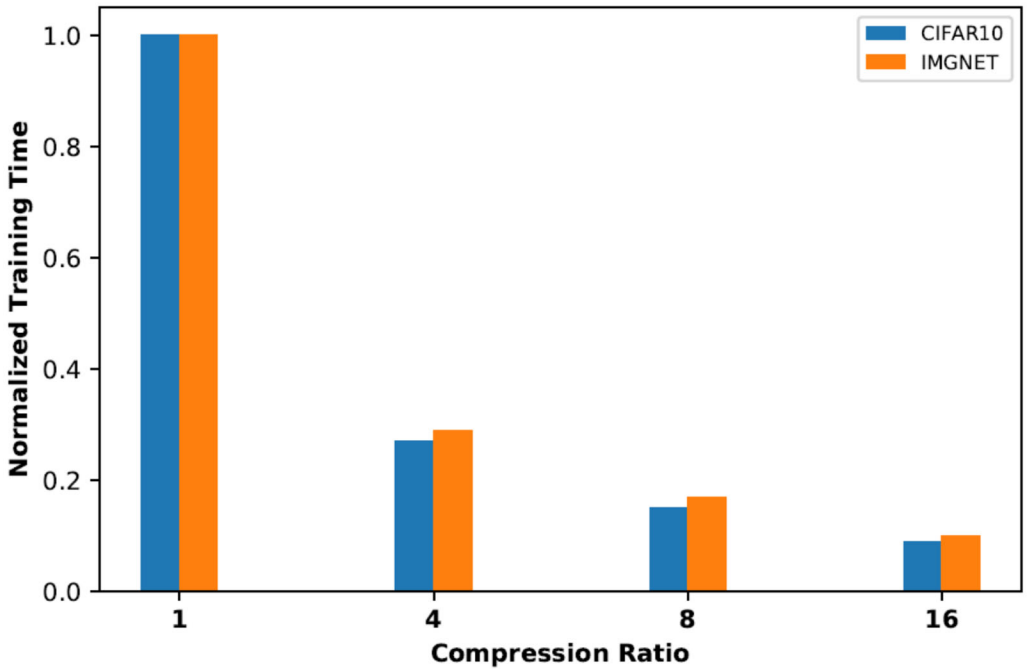
In this part, we analyze the effect on the system performance from the inference error rate in the classification problem and the inference bias in the regression problem. For the simplicity, it is assumed that $K = 1$ in the analysis, which can be extended to the multi-server MEC scenarios because the server selection (CAP association) can be decoupled from the online inference.

Impact of Inference Error Rate of Classification Problem on System Cost:

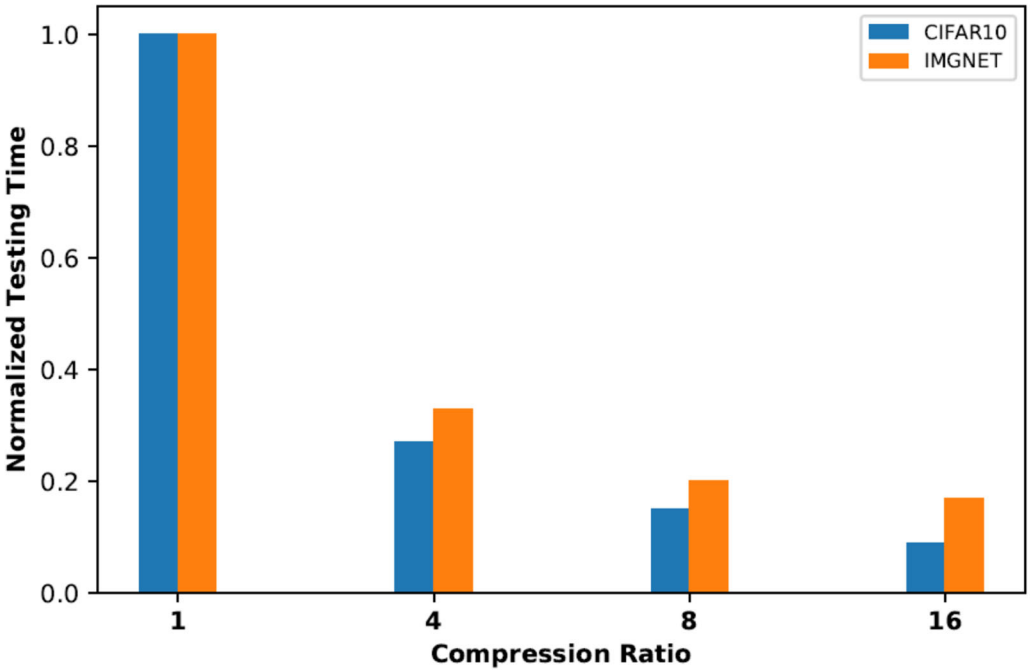
For the multi-class classification problem, we define *the inference error rate as* $\rho = \frac{Number\ of\ false\ predictions}{Total\ number\ of\ predictions}$. To analyze the effect of inference error rate in predicting $\boldsymbol{D}^*$ based on the pre-trained MTFNN model, Observation 1 is first given as follows.

Observation 1 *If error occurs on the prediction of* $\boldsymbol{D}^*$*, the system performance is not related to the prediction of* $\boldsymbol{\Theta}^*$**.**

Based *on Observation 1*, we have the following Theorem.



(a) Comparison of the normalized testing time



(b) Comparison of the normalized training time

Figure 59: Comparison of the normalized testing time and training time of the vanilla models for various compression ratios for CIFAR10 and ImageNet datasets

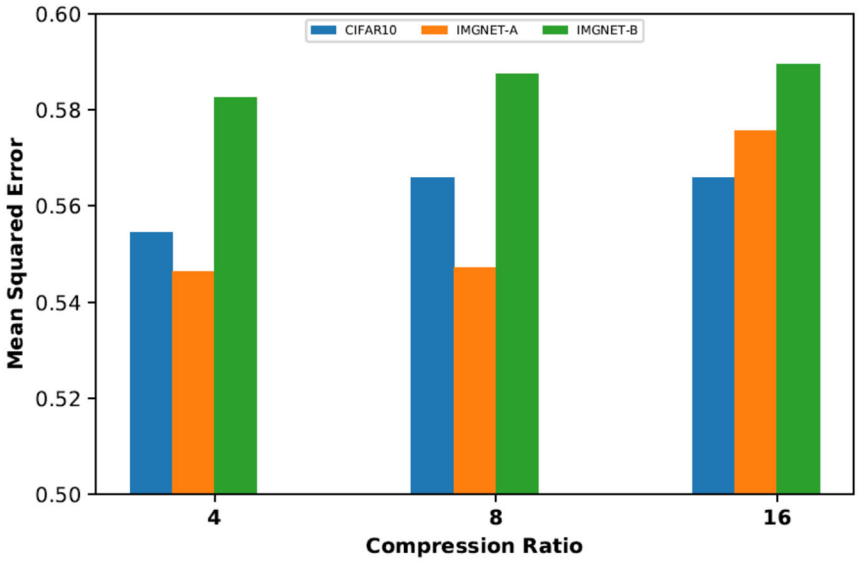


Figure 60: Comparison of the mean squared error of the vanilla models for the original dataset (compres-sion ratio =1) and the compressed dataset (latent variables) with compression ratio=4,8,16.

Theorem 1 *For each MU n, $n \in \mathcal{N}$, let $\Delta_n = |\mathcal{O}_o^n - \mathcal{O}_l^n|$ denoting the difference between the cost using offloading strategy and the cost using local computing. The additional cost introduced by the multi-class classification is $\rho\Delta_n$.*

Proof: For the MU n, the predicted offloading decision can be calculated as

$$D_n^p = \rho\widetilde{D}_n + (1-\rho)D_n, \tag{4.1}$$

where $\widetilde{D}_n$ denotes the opposite relation of D_n, i.e.,

$$\widetilde{D}_n = \begin{cases} 0, & if\ D_n = 1 \\ 1, & if\ D_n = 0 \end{cases} \tag{4.2}$$

For MU n, the optimal cost can be expressed as

$$\mathcal{O}_n = \mathcal{O}_l^n + D_n(\mathcal{O}_o^n - \mathcal{O}_l^n). \tag{4.3}$$

Accordingly, the cost predicted by the pre-trained MTFNN model can be expressed as

$$\mathcal{O}_n^p = \mathcal{O}_l^n + D_n^p(\mathcal{O}_o^n - \mathcal{O}_l^n). \tag{4.4}$$

Substituting (4.1) into (4.4), $\mathcal{O}_n^p$ can be derived as

$$\mathcal{O}_n^p = \mathcal{O}_l^n + D_n(\mathcal{O}_o^n - \mathcal{O}_l^n) + \rho\big(\widetilde{D}_n - D_n\big)(\mathcal{O}_o^n - \mathcal{O}_l^n) \tag{4.5}$$

Substituting (4.3) into (4.5), we have

$$\mathcal{O}_n^p = \mathcal{O}_n + \rho\big(\widetilde{D}_n - D_n\big)(\mathcal{O}_o^n - \mathcal{O}_l^n) \tag{4.6}$$

If $D_n = 1$, i.e., $\mathcal{O}_o^n < \mathcal{O}_l^n$ and $\widetilde{D}_n = 0$ hold, then (4.6) can be further expressed as $\mathcal{O}_n^p = \mathcal{O}_n - \rho(-\Delta_n) = \mathcal{O}_n + \rho\Delta_n$. Otherwise, when $D_n = 0$, i.e., $\mathcal{O}_o^n > \mathcal{O}_l^n$ and $\widetilde{D}_n = 1$ hold. As a result, (4.6) can be further expressed as $\mathcal{O}_n^p = \mathcal{O}_n + \rho\Delta_n$. To summarize, the additional over-head introduced by the multi-class classification in the pre-trained MTFNN model is $\rho\Delta_n$.

Impact of Inference Bias on System Cost:

For the regression problem, we define the inference bias of MU *n* as $b_n = \widetilde{\Theta}_n - \Theta_n$, where Θ_n is the ground truth of resource allocation ratio, $\widetilde{\Theta}_n$ denotes the predicted resource allocation ratio. To analyze the effect of inference bias in predicting Θ^* based on the pre-trained MTFNN model, *Observation 2* is presented as follows.

Observation 2 *Suppose no error occurs on the prediction of D^*, the system performance is only affected by the inference bias*.

Based on *Observation 2*, we have the following Theorem.

Theorem 2 *Let $\omega_n = \frac{D_n c_n}{f_n \Theta_n}(\alpha + (1-\alpha)P_l^n)$, the additional cost introduced by the regression is $\omega_n / \left|1 + \frac{\Theta_n}{b_n}\right|$.*

Proof: Suppose that no error occurs during the multi-class classification, i.e., $\rho = 0$. Considering that the predicted resource allocation ratio $\left(\widetilde{\Theta}_n\right)$ affects the offloading cost, then we have

$$\mathcal{O}_n^p = \mathcal{O}_l^n + D_n\left(\tilde{\mathcal{O}}_o^n - \mathcal{O}_l^n\right), \tag{4.7}$$

where $\tilde{\mathcal{O}}_o^n$ denotes the offloading cost based on the inference results given by the pre-trained MTFNN model.

Based on (4.3) and (4.7), we have

$$\left|\mathcal{O}_n^p - \mathcal{O}_n\right| = D_n \left|\tilde{\mathcal{O}}_o^n - \mathcal{O}_o^n\right|. \tag{4.8}$$

Substituting (3.23) and (3.24) into (3.25), $\mathcal{O}_o^n$ can be rewritten as

$$\mathcal{O}_o^n = \frac{s_n}{r_u^n}\left(\alpha + (1-\alpha)P_{t,n}^i\right) + \frac{c_n}{F\Theta_n}(\alpha + (1-\alpha)P_l^n) + \frac{\omega_n}{r_d^n}\left(\alpha + (1-\alpha)P_{d,n}^i\right), \tag{4.9}$$

where F denotes the total computational resources of the CAP. $P_{t,n}^i$ and $P_{d,n}^i$ denote the power consumption of MU n uploading data to the CAP and downloading the execution result from CAP via the sub-band i, respectively.

Accordingly, $\tilde{\mathcal{O}}_o^n$ can be rewritten as

$$\tilde{\mathcal{O}}_o^n = \frac{s_n}{r_u^n}\left(\alpha + (1-\alpha)P_{t,n}^i\right) + \frac{c_n}{F\Theta_n}(\alpha + (1-\alpha)P_l^n) + \frac{\omega_n}{r_d^n}\left(\alpha + (1-\alpha)P_{d,n}^i\right), \tag{4.10}$$

Substituting (4.9) and (4.10) into (4.8), we can get

$$\left|\mathcal{O}_n^p - \mathcal{O}_n\right| = \frac{D_n c_n}{F}(\alpha + (1-\alpha)P_l^n)\left|\frac{1}{\widetilde{\Theta}_n} - \frac{1}{\Theta_n}\right| \tag{4.11}$$

Considering that b_n denotes the inference bias of the regression, then $\widetilde{\Theta}_n = \Theta_n + b_n$ holds. As a result, (4.11) can be rewritten as

$$\left|\mathcal{O}_n^p - \mathcal{O}_n\right| = \frac{D_n c_n}{F}(\alpha + (1-\alpha)P_l^n)\left|\frac{1}{\Theta_n + b_n} - \frac{1}{\Theta_n}\right| \tag{4-12}$$

$$= \frac{D_n c_n}{f_n \Theta_n} (\alpha + (1-\alpha) P_I^n) \left| \frac{1}{\frac{\Theta_n}{b_n} + 1} \right|$$

This ends the proof.

According to *Theorem 1* and *Theorem 2*, the following Corollary is given to illustrate the total cost of MU *n* introduced by the pre-trained MTFNN model, i.e., C_n.

Corollary 1 Due to the imperfectness of the pre-trained MTFNN model, i.e., there exist inference error rate in multi-class classification ($\rho \neq 0$) and inference bias in regression ($b_n \neq 0$), the total cost introduced by the pre-trained MTFNN model is obtained as

$$C_n = \rho \Delta_n + (1-\rho)\omega_n / \left| 1 + \frac{\Theta_n}{b_n} \right| \tag{4.13}$$

Remark 4 *Given the MUs' profile information, it is observed from (4.13) that* $\boldsymbol{C}_n$ *is proportional to* ρ *and* b_n*. Note that* $\boldsymbol{C}_n$ *is dominated by the classification problem if* ρ *is relatively large (e.g.,* $\rho \longrightarrow 1$ *in one extreme case). Only when* ρ *is relatively small (e.g.,* $\rho \longrightarrow 0$ *in another extreme case),* $\boldsymbol{C}_n$ *is mainly determined by the regression bias.*

4.2.2.2 Implementation of the Pre-trained MTFNN Model

The implementation of the pre-trained MTFNN model is illustrated in Fig. 61. Due to the diversity of the MUs' input profile, the MTFNN model needs to be pretrained for each scenario, in which there exist a different number of MUs offloading jobs to the CAP. After the offline training, the pre-trained MTFNN models corresponding to each scenario can be stored at the CAP in advance. Given a set of input parameters (e.g., number of MUs), the CAP selects the pre-trained MTFNN model accordingly (i.e., Phase 1 in Fig. 61) and then performs the on-line inference (i.e., Phase 2 in Fig. 61). Moreover, the size of the pre-trained MTFNN model for each scenario is less than 2

KB, which indicates that the MES at the CAP has enough storage space to save the pre-trained MTFNN models[4].

4.2.2.3 Medium Access Control (MAC) Protocol

Different from using time-division multiple access (TDMA) way for the uplink offloading in [143, 295], in our offloading framework, the time is divided into multiple frames, and it is assumed that the channel remains static within each frame and the system is synchronized using Beacon. Here, we exemplify the proposed MAC protocol for each CAP within a single frame. Each frame is sub-divided into two non-overlapping periods, i.e., learning period (LP) and offloading period (OP), as illustrated in Fig. 62, where the pre-trained MTFNN model can be deployed at the CAP [296, 297]. During the LP, all MUs with offloading jobs send a "Request" message to CAP in an OMA (e.g., OFDMA) or an advanced NOMA way, which contains the parameters input to the MTFNN model. On receiving the "Request", CAP decodes the "Request" message and then infers the $\boldsymbol{D}^*$ and $\boldsymbol{\Theta}^*$ based on the selected MTFNN model file. Then, the CAP notifies the MUs of the predicted $\boldsymbol{D}^*$ and $\boldsymbol{\Theta}^*$ by replying a "Response" message after short interframe space (SIFS). Due to the relatively small value of inference time (e.g., less than $4\mu s$ for $N = 8$) compared with the SIFS (e.g., $16\mu s$ in IEEE 802.11a), so it is approximately considered that the offloading strategy is made within SIFS by the CAP. During the OP, the MUs with $\boldsymbol{D}^* \neq 0$ upload the jobs data to CAP. Once the data is received, the CAP processes the jobs based on the allocated computational resources. After all the jobs have been processed, the CAP returns the executed results to the MUs. Therefore, all the offloading jobs are received and computed within one frame, and then in next frame the same procedure is performed for new set of offloading requests.

[4] When the number of mobile devices offloading jobs to the same CAP simultaneously is generally not too large [294], e.g., less than 20, the total space storing the pre-trained MTFNN models is less than 40 KB, which is generally far below the capacity of the CAP. In such case, the pre-trained MTFNN models can be cached into the memory of MES at the CAP in advance to perform the inference more efficiently.

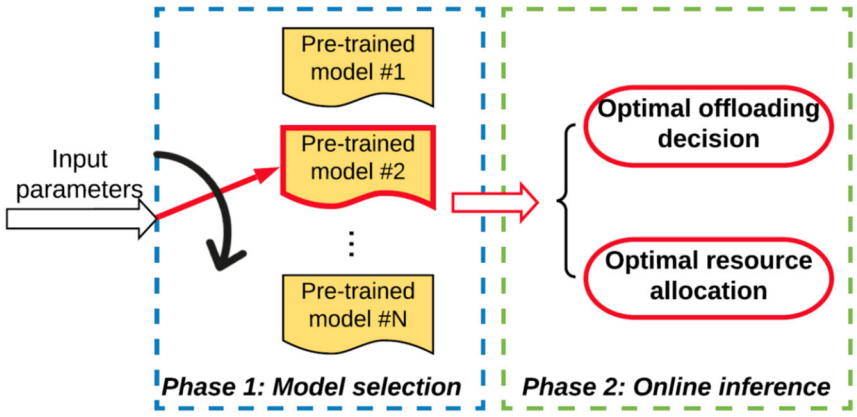


Figure 61: Implementation of the MTFNN prediction online

Signaling Overhead Discussion:

In our proposed MTFNN based offloading protocol, the CAP needs to obtain the details of computation jobs (i.e., some parameters of the jobs, which mainly includes the data size (s), the CPU resource (c), the bandwidth (W), coefficient (α), the MU's CPU capability (f_l) and transmission power (P_t)) from each MU to predict the optimal offloading strategy. However, the signaling overhead is not a big concern due to the following two reasons.

- Firstly, *these parameters can be piggybacked into the "Request" packet. The size of "Request" packet is not too large and can be transmitted simultaneously.* Owing that the "Request" packet of each MU is transmitted to the CAP simultaneously, so the time overhead to transmit the "Request" packet can be decreased to a large extent.
- Secondly, *some of these parameters may not change within a period of time, which can help to further decrease signaling overhead*. Specifically, some of the parameters such as f_l, P_t, W and α may keep unchanged in a period of time, and even the remaining parameters (i.e., s and c) may also be almost unchanged if MU's input data source keeps stable. In this case, the time overhead can be decreased significantly. To go a step further, if the channel condition keeps unchanged (e.g., the MUs are not moving and the channel between MUs and AP is line-of-sight (LOS)), the signaling may occur only once because the predicted offloading strategy is valid for an extended period.

Suppose that given a distributed approach, even though the MU can make offloading decisions (i.e., processing job locally or offloading to the CAP) in the distributed approaches, their offloading decisions can hardly be the optimal due to the computation jobs information are not shared. More importantly, the CAP can also hardly allocate the computation resources efficiently using distributed approaches if the CAP does not know the details of computation jobs of all MUs.

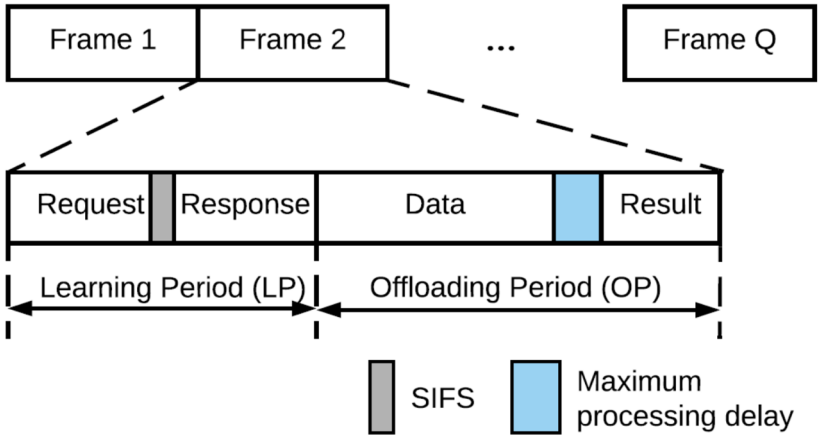


Figure 62: MAC protocol for the offloading between MUs and CAP

4.2.2.4 Testing Results

During the testing phase, the performance of the MTFNN model is evaluated based on the outputs and the corresponding labels[5]. To demonstrate the superiority on resolving the MINLP problem using the proposed MTFNN model, we compare with a benchmark scheme, spatial branch and bound (sBB), which is implemented using the MATLAB toolbox of the APMonitor Optimization Suite [298]. It is worth noting that for the fairness of comparison, the performance of online inference of both sBB method and the proposed MTFNN model are compared on the same system with Intel Xeon(R) CPU E5-2650 @ 2.0 GHz ($\times$ 16) processor[6]. In the following, we introduce two testing indexes.

[5] The outputs obtained from the MTFNN model are performed 50 epochs and normalized to make sure the condition C4 is met.

[6] This means that even though the MTFNN model can be trained offline via GPU, however, the online inference is still performed via CPU.

- Computation complexity. We evaluate the complexity with the execution time per sample, which is defined as $t = \frac{Total\ execution\ time}{Number\ of\ samples}$. We keep a record of time cost to solve the original optimization problem **P1** using sBB algorithm and the pre-trained MTFNN model. Note that the offline training needs to be performed only once, the execution time denotes the inference time.
- Computation accuracy. We define the accuracy of the offloading decision inference in the multiclass classification problem as $\eta = \frac{Number\ of\ correct\ predictions}{Total\ number\ of\ predictions}$. Denote m as the total number of samples, N is the total number of devices, we use the MSE function to indicate the accuracy of resource allocation strategy in the regression problem, i.e., $\varepsilon = \frac{1}{mN}\sum_{i=1}^{m}\sum_{j=1}^{N}\left(y_j^i - x_j^i\right)^2$, where y_j^i is the predicted value of d_j from the i-th sample and x_j^i is its label.

Comparison of Computation Complexity:

The comparison of computation complexity between the sBB algorithm and MTFNN model with $\chi_c = \chi_r = 1$ is compared in Table 9[7]. It can be observed that compared to the sBB scheme, the pre-trained MTFNN model obtains much lower complexity on the premise of a relatively high accuracy, and the time cost to solve the problem **P1** using MTFNN model is even less than one-thousandth of the sBB scheme[8]. The main reasons are as follows. As the number of MUs (N) grows, the conventional exhaustive search strategy suffers from the exponential time complexity $O((2g)^N)$, where $g = \frac{1}{\omega} + 1$, $\omega \in (0,1)$ denotes the granularity of computational resource allocation. Meanwhile, to solve the MINLP problem, the sBB always has exponential worst-case complexity, i.e., $O(2^N)$ [299]. In the pre-trained MTFNN model, the quadratic time complexity can be achieved as $O(M^2L)$, where L is the number of layers, M is the number of neurons in a hidden layer which indicates the scale of the neuron network model. Besides, we only need to train our learning model once, which can be performed offline via the machines with strong computing and

[7] Note that the "Schemes" is abbreviated as \S" to save space in Table 4.4 and Table 4.5.

[8] Note that different from the sBB scheme where the inference time t keeps increasing as the increase of number of MUs N, by using the pre-trained MTFNN model, the inference time t is not increased with N. This is because the testing index t denotes the execution time per sample. As the total number of collected data samples increases while the total inference time grows not too much, so the average inference time per sample is not increased with N.

storage capabilities, e.g., the GPU clusters. Therefore, the proposed MTFNN method has a relatively low complexity compared to the sBB and exhaustive search schemes.

Table 9: Results of MTFNN with $\boldsymbol{\chi_c = \chi_r = 1}$

$N\eta, \varepsilon, tS$	sBB	MTFNN with $\chi_c = \chi_r = 1$
2	70%, 0.055, 14.1 ms	96%, 0.016, 2.5 μs
3	62%, 0.047, 14.2 ms	89%, 0.027, 2.5 μs
4	58%, 0.053, 14.5 ms	83%, 0.029, 2.2 μs
mygray height5	47%, 0.051, 15.2 ms	50%, 0.021, 2.0 μs

Comparison of Computation Accuracy:

The comparison of computation accuracy in terms of η and ε is given in Table 9, where the MTFNN model is with $\chi_c = \chi_r = 1$. It can be seen that MTFNN model obtains a relatively high accuracy on the premise of much lower complexity. Specifically, the MTFNN model outperforms the sBB by average 40% in the classification inference accuracy and the MSE of MTFNN is about only a half of the sBB scheme. To demonstrate the computation accuracy of the regression problem, the comparison of regression inference for the case of three devices is presented in Fig. 63. It can be observed that the pre-trained MTFNN model outperforms the sBB algorithm, which matches the ground truth well. The reason why the proposed MTFNN model outperforms the other method is that the MTFNN model is trained offline with the data set collected using the exhaustive searching method, which can always obtain the optimal solutions to the formulated MINLP problem. In other words, the labeled data is with relatively high quality and thus the NN model can perform well if it is trained with appropriate hyper-parameters and the number of feeding data samples is large enough.

Nevertheless, it is observed from Table 9 that when N = 5, the inference accuracy of offloading decision given by the MTFNN model with $\chi_c = \chi_r = 1$ severely decreases. The main reason behind this is that the complexity of the multi-class classification problem grows exponentially with the increase of N. The higher the complexity the lower accuracy for offloading decision, and the system performance is currently dominated by the inference error rate of classification, as illustrated in *Remark 3*. Therefore, in order to keep a relatively high inference accuracy in

different scenarios[9], the weight of loss function in the MTFNN model should be adjusted based on N.

Impact of the Weights of Loss Function on Inference Accuracy:

Note that in Table 9, the weights of loss functions ($\boldsymbol{l}_c$ and $\boldsymbol{l}_r$) are set to be identical, i.e., $\chi_c = \chi_r = 1$. In this case, the MTFNN method significantly outperforms the sBB scheme in the cases of $N \leq 4$. With the increase of N, it is known that the complexity of the regression problem increases linearly while the complexity of the multi-class classification problem grows exponetially.

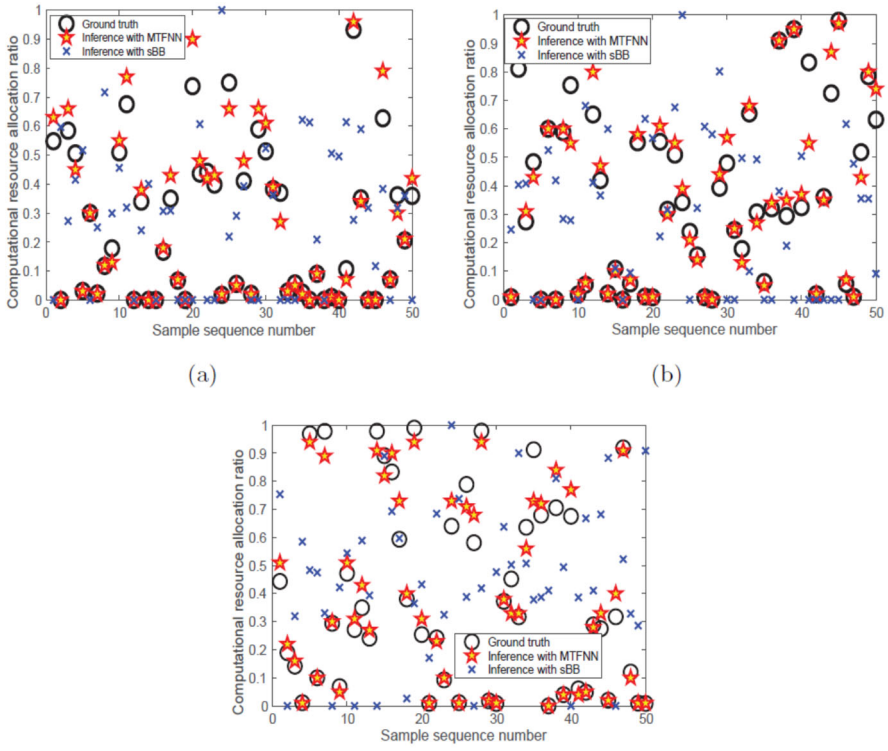


Figure 63: The computational resource ratio (i.e., $\Theta = [\Theta_1, \Theta_2, \Theta_3]$) predicted by the pretrained MTFNN model with $\chi_c = \chi_r = 1$, where the number of MUs is 3. $\Theta_1, \Theta_2,$ and Θ_3 is respectively shown in (a), (b) and (c)

In such a situation, the inference of classification problem deteriorates the overall performance of multi-task learning due to the relatively high complexity. In order to resolve this issue, we adjust the weights of the loss function according to the value of N. When N is not too large, e.g., $N \leq 4$

[9] Different scenarios mean that there exist a different number of devices offloading jobs to the MES at the same time.

in the small-scale network, the weight for classification and regression is prone to be the same, e.g., $\chi_c = \chi_r = 1$. This can help to ensure the accuracy of both classification and regression. When N continues to become large, e.g., $N \geq 5$ in the larger-scale network, the weight of classification should be reduced while the weight of regression should be improved accordingly. Here, we set $\chi_c = 0$ and $\chi_r = 1$. At this time, *the multi-task learning problem degenerates into a pure regression problem*, and the output of the MTFNN model only indicates the resource allocation vector $\boldsymbol{\Theta}^*$. In order to obtain the offloading decision vector $\boldsymbol{D}^*$, we set a threshold Θ_{th}. If $\Theta_{\text{n}} \geq \Theta_{\text{th}}$, we have $D_n = 1$, otherwise $D_n = 0$. According to the ground truth in the training dataset, Θ_{th} is set to be 0:15 in Table 10. It is observed from Table 10 that the accuracy of the sBB scheme has been steadily dropping and the MSE is increased as well. In contrast, the inference accuracy of the MTFNN model with $\chi_c = 0$ and $\chi_r = 1$ always outperforms the sBB scheme and keeps steady. Compared to MTFNN model with $\chi_c = \chi_r = 1$, the MSE of the regression given by the MTFNN model with $\chi_c = 0$ and $\chi_r = 1$ is further decreased, which is only about 15% of the sBB scheme when $N = 5$.

Table 10: Results of MTFNN with $\boldsymbol{\chi_c = \chi_r = 1}$

$N\eta$, ε, tS	sBB	MTFNN with $\chi c = 0$, χr
mygray height5	47%, 0.051, 15.2 *ms*	78%, 0.009, 5.0 μs
6	42%, 0.092, 15.8 *ms*	82%, 0.009, 5.7 μs
7	38%, 0.095, 16.6 *ms*	81%, 0.009, 3.6 μs
8	34%, 0.097, 16.9 *ms*	78%, 0.009, 3.9 μs

Impact of the Number of Training Samples on Inference Accuracy:

From Fig. 64(a)-(c), it is observed that the accuracy of offloading decision is low for a small number of training samples while increases gradually with the number of training samples, and the change of MSE of regression shows an opposite trend. This is because that for a small number of training examples, the MTFNN model can only learn very few distinguishing features and the learned features are not sufficient enough to perform inference accurately. With the increase of training samples, the MTFNN model gradually extracts more abstract features from the data, which allow the MTFNN model to learn more about the relationship between the inputs and outputs of

the formulated MINLP problem. Therefore, the inference accuracy keeps increasing with the training examples and then starts to saturate after there are enough training examples. Moreover, it can be observed that when the number of MUs (N) becomes larger, more training samples are required for the MTFNN model to achieve similar accuracy.

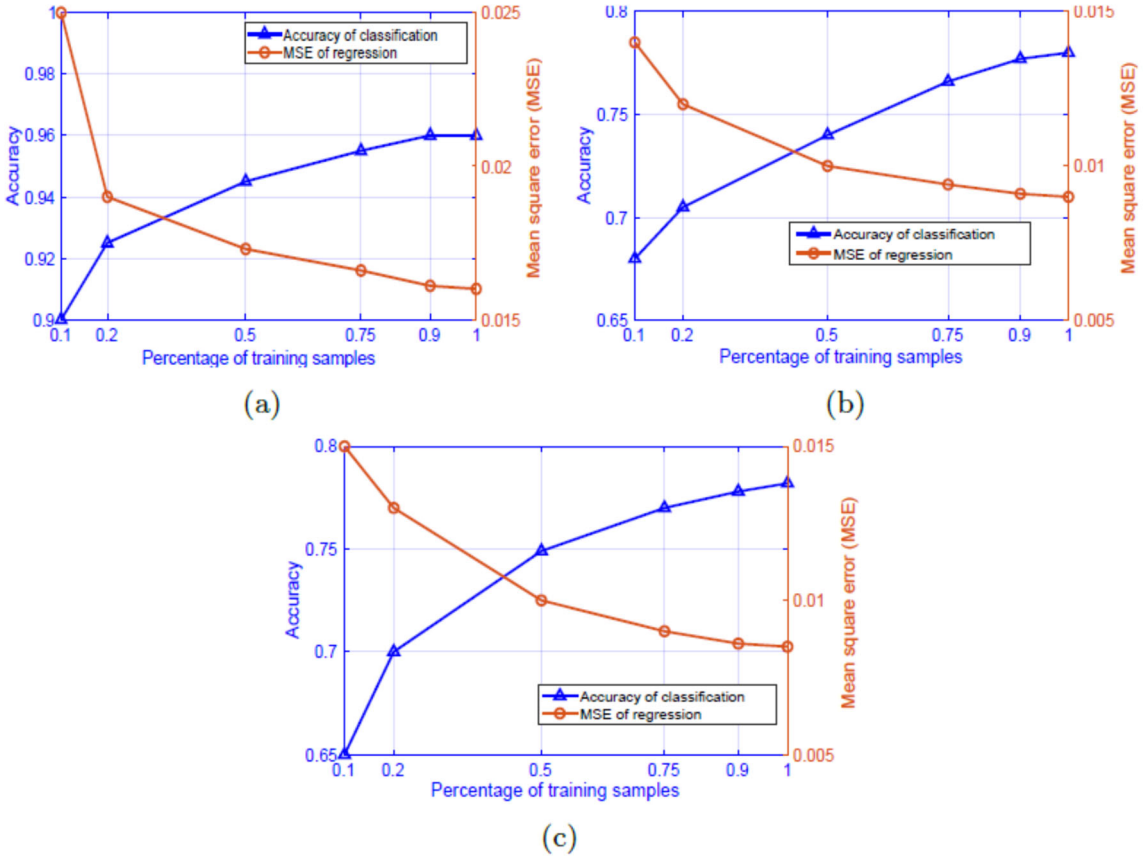


Figure 64: The accuracy of the classification (η) and the mean square error (MSE) of the regression (ε) for $N = 2$, $N = 5$ and $N = 8$, is respectively shown in (a), (b) and (c). In (a), the number of total training samples is 3.2×10^4, and $\chi_c = \chi_r = 1$. In (b) and (c), the number of total training samples is 8×10^4 and 2.4×10^5, respectively, and$\chi_c = 0$, $\chi_r = 1$.

4.2.2.5 Performance Evaluation of MTFNN based Offloading Protocol

In our simulations, to evaluate the effectiveness of our proposed MTFNN model based offloading scheme (MOF), we consider the scenario with a CAP surrounded by N MUs randomly scattered within 200 m. Four benchmark offloading approaches are introduced as follows.

- **Full offloading (FOF)**. The FOF scheme denotes that all the jobs will be executed at the CAP totally and the whole computational resource (F) is allocated equally to each device, i.e., $D_n = 1$, and $f_n = F/N, \ \forall n \in \mathcal{N}$.
- **None offloading (NOF)**. The NOF scheme denotes that all the jobs will be executed locally by the MUs themselves, i.e., $D_n = 1, \forall n \in \mathcal{N}$.
- **Random offloading (ROF)**. The ROF scheme denotes that all the jobs will be executed by the two ways above randomly. Without loss of generality, the simulation is performed for an average of 1000 runs.
- **sBB based offloading (SOF)**. The SOF scheme denotes that all the jobs will be executed according to the offloading strategy given by the sBB algorithm. We compare the proposed MOF scheme with the other four methods with respect to the total system cost (i.e., $\mathcal{O}_{\text{total}}$). The CPU frequency of each MU is $f_l = 0.5$ GHz, the coefficient of each MU is set to be $\alpha = 0.5$ for simplicity. Some other parameters can refer to Section 3.2.3.2.3.

Results and Discussions:

Fig. 65(a) presents the impact of the number of devices (N) on the total system cost of the MEC network, where the size of the input data is 200 kbits. Taken as a whole, the system cost of all the schemes increase by nature when N keeps increasing. Specifically, the "NOF" scheme is with the highest system cost when $N \leq 4$ while the "FOF" scheme becomes the highest instead when $N > 4$. This is because that when N is not too large, compared with offloading jobs to the CAP which has enough computational resource to meet the offloading demands, the MUs consume much more energy locally. As N keeps increasing, the computational capability and storage capacity of the CAP is not sufficient enough to meet the offloading demands from all devices. Due to the inference accuracy of "SOF" scheme drops rapidly with increasing of N (as illustrated in Table 10), the "SOF" only performs well when N is relatively small (e.g., $N \leq 3$). Fortunately, it is observed that the proposed "MOF" scheme always outperforms the benchmark schemes because it can make an optimal offloading strategy in most cases by performing the inference online. This validates the effectiveness of the proposed MTFNN model and also indicates that the intelligent offloading decision making plays an increasingly important role in the MEC networks.

Fig. 65(b) shows the system cost with respect to the increasing input data size of the offloading job, where N = 3. It is obvious that the job with bigger data size requires more time and energy consumption. As a result, the system cost increases with the increasing data size. As we expected, the curve of the "NOF" scheme is with the fastest growth as the data size is increased. This indicates that the larger job size the more energy consumption of the MUs.

The system cost with an increasing computation capability of MES (denoted as F) is given in Fig. 65(c), where $N = 3$ and the job input size is 200 kbits. It is observed that with the increase of F, except for the "NOF" scheme, the system cost of the other three offloading schemes decrease significantly. This is because that the "NOF" scheme computes the job locally, so the amount of computational resource owned by the CAP does not affect the system cost using "NOF" scheme. In contrast, as the computation capability of the CAP increases, the jobs execution time of the other four schemes can be shortened because more computation resources will be allocated accordingly. Furthermore, with the increase of F, "processing jobs via offloading" gradually becomes the optimal strategy to minimize the time cost. Therefore, it can be seen that the system cost of "FOF" and "MOF" schemes are almost the same when F = 2:5 GHz.

4.3 Knowledge Extraction using Machine Learning and Deep Learning

4.3.1 Evidence Theory for Big Data Processing

4.3.1.1 Data Collection

In our experimental data collection, we used the proposed platform ([202]) in Figure 22 to monitor the air quality inside an apartment during summer season. We put five sets of Sensordrone nodes and Android© smartphones with related apps in different parts of the apartment named room1, room2, living, dining, and kitchen, respectively. Sensing interval can be set arbitrarily such as 1, 5, or 10 seconds. Smaller interval is suitable to have more samples for accurate monitoring. Because we wanted to monitor environment for at least 12 hours without recharging sensor nodes and with reasonable accuracy, so for our data collection we set interval time to five second for all five sensor nodes. The sensor nodes measured temperature, humidity, pressure, carbon monoxide, and battery level of sensor node. In addition, time stamps and GPS location data are uploaded to the server. Here only temperature and humidity data used for our case study in this work.

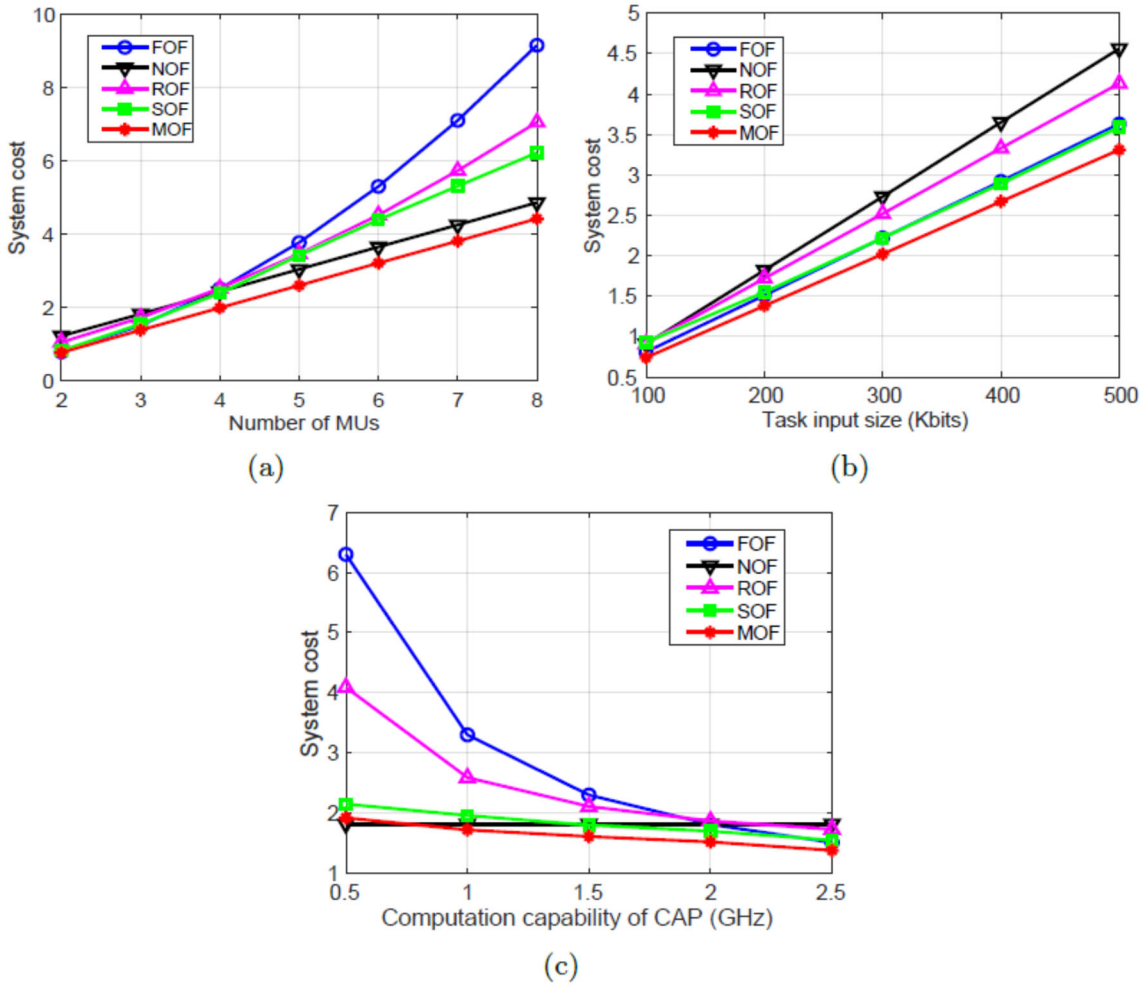


Figure 65: In (a), system cost versus the number of MUs is shown, where task input size is 200 kbits. System cost versus the task input size is shown in (b), where $\boldsymbol{N = 3}$. System cost versus the computation capability of the CAP is shown in (c), where $\boldsymbol{N = 3}$ and task input size is 200 kbits.

The monitored temperature and humidity data for ten-hours sensing period including 7500 data samples and their mappings in comfort zone graph are shown in Figure 66. Fluctuations in temperature and humidity are caused by turning on the air conditioner (AC) periodically for cooling during the summer. AC set to 77 degree Fahrenheit. It turned off for the last four hours. Then temperature and humidity started to increase in all places as expected.

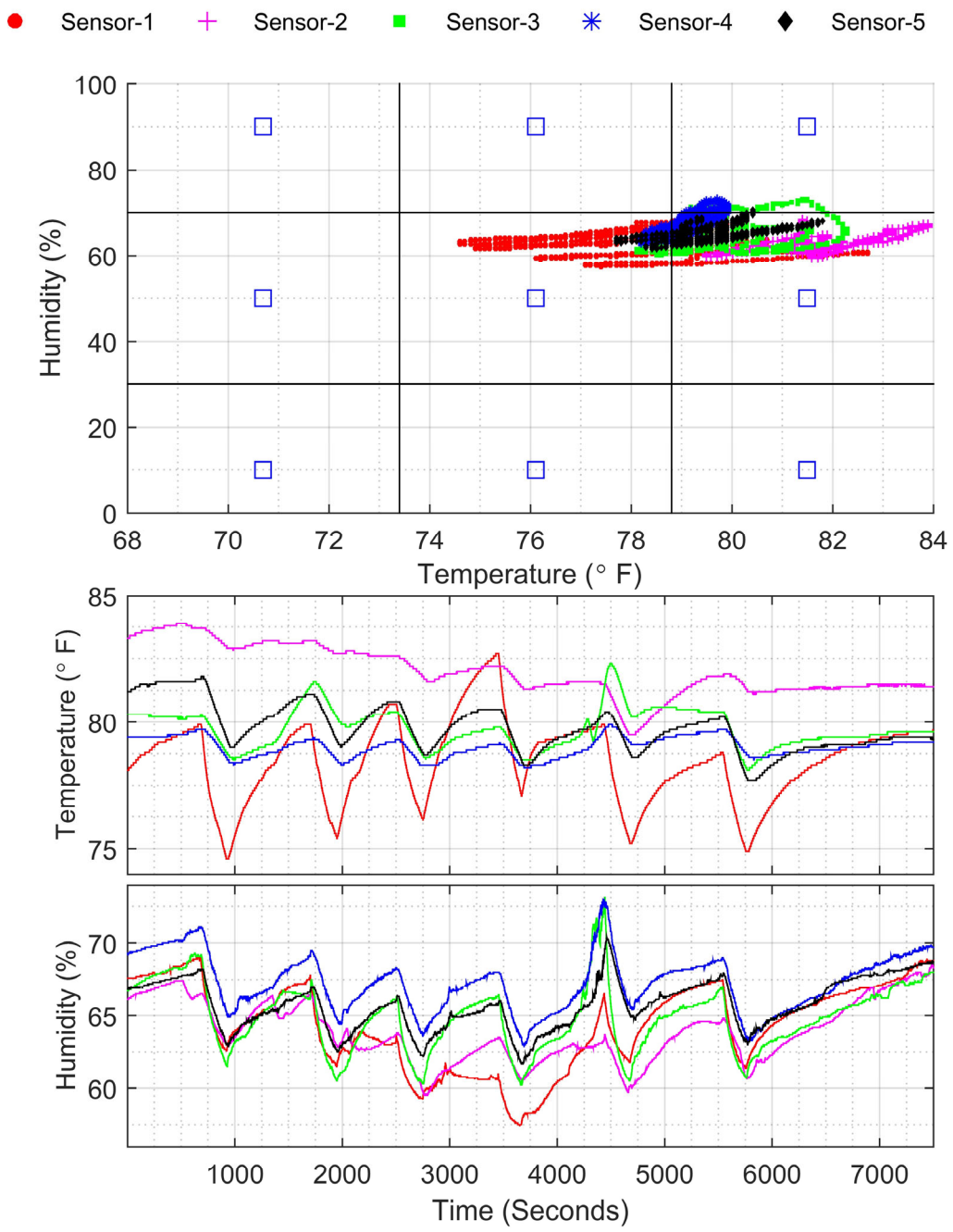


Figure 66: Temperature, Humidity data and related Comfort zone

4.3.1.2 Data Analysis and Decision Making

Implementation of non-overlapped model This section explains the details of our proposed model and implementation of DST and DSmT related combination rules based on our model. Both simulation results and real data analysis based on the experiments are shown to determine the comfort zone inside the apartment. According to the "Comfort Zone" in ISO7730-1984 standard ASHRAE shown in Figure 21, ([300]) defined 9 hypotheses/zones including the comfort zone and 8 other zones around the comfort zone. We will call this model the non-overlapped model. Figure 67 shows the 9 zones for the summer season. In Figure 64, small blue square markers show the center of related zones and red solid lines are used as a boundary to distinguish between different hypotheses. Table 11 displays temperature and humidity values and feeling definition for related zones based on the non-overlapped model. Thus the frame of discernment for feeling zone evaluation is:

$$\Theta = \{l_1 = \mathrm{I}, l_2 = \mathrm{II}, \dots, l_9 = "IX"\} \tag{4.14}$$

Here "I" refers to the first hypothesis and "II" refers to the second hypothesis and so on. Because all 9 hypotheses are exclusive and exhaustive, it satisfies the Dempster-Shafer model. It is noted that uncertainty is not considered in this model.

Feeling definitions in Table 11 based on Figure 67 explain the human feeling for the range of temperature and humidity in each zones. For example, zone "I" refers to "too cold and humid" and so on. Based on our proposed method in Figure 67, Dempster's rule of combination can be applied to our data to compute total mass. Because in this model all 9 hypotheses are singleton and exclusive, the total mass and belief functions are equal.

In order to calculate the mass function, we first calculate the normalized Euclidean distance between measured data from sensors and hypotheses parameters:

$$d_i^{l_j} = \left(\sum_{x=1}^{m} \left(\frac{S_x - f_x^{l_j}}{f_{max} - f_{min}} \right)^2 \right)^{1/2} \tag{4.15}$$

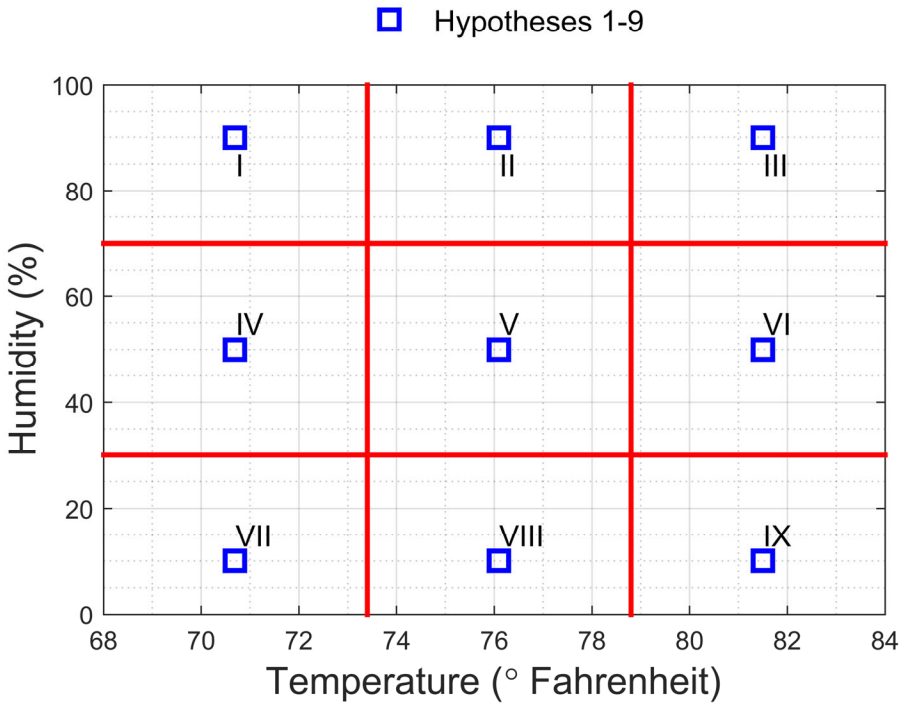


Figure 67: Proposed 9 hypotheses model for summer season (Non-overlapped model)

Here $d_i^{l_j}$ refers to the distance between sensor i and hypothesis j, S_x is sensor data and $f_x^{l_j}$ is the reference value of hypothesis j. $f_{max} - f_{min}$ is used for normalization. Dimensionality is shown by m. In our proposed models $m = 2$, to represent Temperature and Humidity. So it simplifies to:

$$d_i^{l_j} = \left(\left(\frac{T_i - T^{l_j}}{T_{max} - T_{min}} \right)^2 + \left(\frac{H_i - H^{l_j}}{H_{max} - H_{min}} \right)^2 \right)^{\frac{1}{2}} \tag{4.16}$$

Then for every sensor node i, distance values related to all hypotheses can be obtained:

$$D_i = \{ d_i^{l_1}, d_i^{l_2}, \dots, d_i^{l_n} \} \tag{4.14}$$

For the small value of distance $d_i^{l_j}$, the probability that the sensor i is belong to zone l_j is higher. Then mass function calculated based on the distance values for each sensor node i related to each hypothesis j:

$$m_i(l_j) = \frac{1/d_i^{l_j}}{\sum_{j=1}^{n}\left(1/d_i^{l_j}\right)} \tag{4.18}$$

Finally mass functions for each sensor node i related to all n hypotheses are:

$$m_i = \{m_i(l_1), m_i(l_2), \ldots, m_i(l_n)\} \tag{4.19}$$

Table 11: The proposed 9 hypotheses model details

Class	Temperature	Humidity	Feeling Definition
I	70.7	90	Too Cold & Humid
II	76.1	90	Too Humid
III	81.5	90	Too Warm & Humid
IV	70.7	50	Too Cold
V	76.1	50	Comfort Zone
VI	81.5	50	Too Warm
VII	70.7	10	Too Cold & Dry
VIII	76.1	10	Too Dry
IX	81.5	10	Too Warm & Dry

Simulation and real data analysis of non-overlapped model:

To validate our proposed model with synthetic data, we generate several random test data sets and feed them as input to our MATLAB© program to calculate the Dempster-Shafer combination based on Figure 60. One of the test data set is shown in Figure 68. Eight set of random data are chosen that move diagonally from bottom left to right top along the time. First two sets are inside zone seven (VII), next set inside zone four (IV), next three sets are inside comfort zone and the last two sets are in zone three (III). The total mass function and related decision based on the maximum values of mass are also shown. Based on the non-overlapped model, all nine hypotheses are singleton and exclusive, so the mass and belief are equal. Although the value of plausibility is greater than mass value (small uncertainty value as an offset), the overall decision result are same for mass, belief, plausibility and pignistic probability, as expected.

Similarly, Figure 69 shows the total mass and the related decision results based on experimental real data. It is observed that, when AC is turned on, six times it moves inside the comfort zone (zone 5) from zone 6. Non-overlapped model is not accurate enough, because it does not consider

high granularities. So as shown in Figure 69, It only could detect four of them. Figure 69 shows that the conflict value during all ten hours are high. Thus it is clear that to overcome the effect of high conflict we need to apply DSmT.

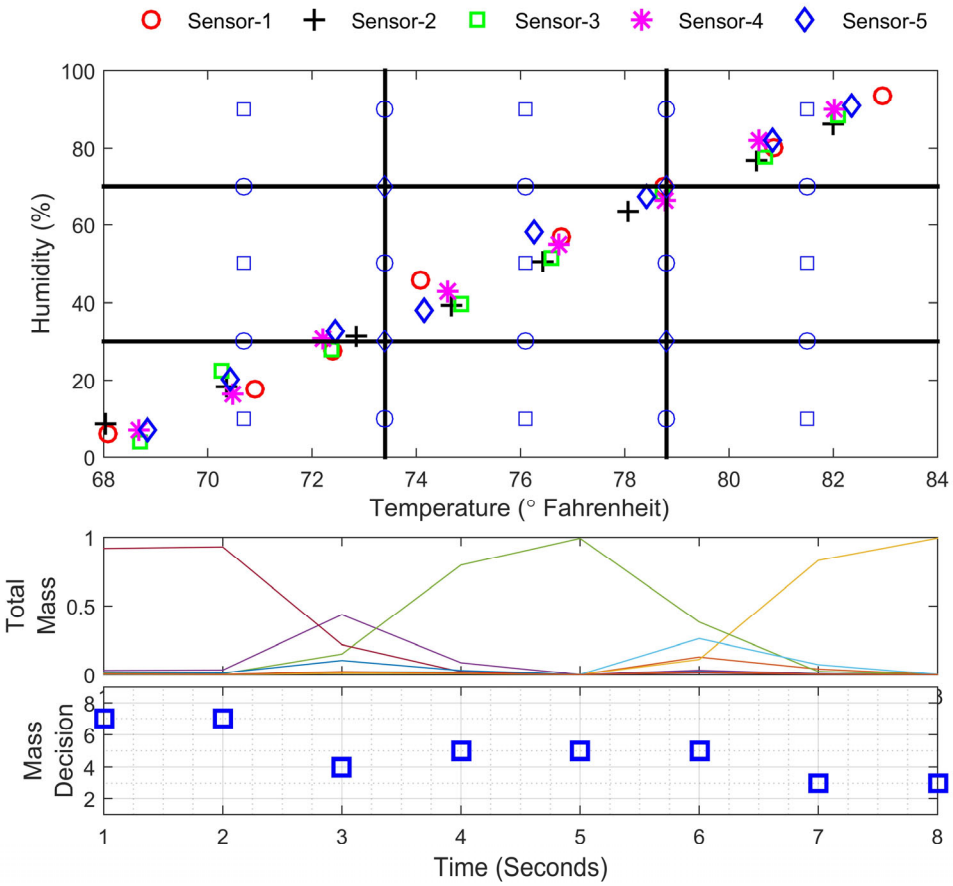


Figure 68: Mass-decision for diagonal test data set (9 hypotheses DST model)

Implementation of overlapped model:

According to the Sensordrone specification document, accuracy of temperature sensors are $+/-0.5°C$ or $+/-0.9°F$ and accuracy of humidity sensors are $+/-$ 2%RH. Therefore measurements reported by Sensordrone sensors add uncertainty factor based on the accuracy range of related sensors. Thus we expand our non-overlapped model to a more accurate one as shown in Figure 70. Dashed lines in Figure 70 are drawn around solid line, intersection between zones, based on $+/-0.9°F$ and $+/-$ 2%RH measurement error for temperature and humidity sensors, respectively. That means each hypotheses can be extended from its solid

line boundary to neighbor dashed line. We call this proposed model the overlapped model. We can treat this new proposed model in two different ways, refined Dempster-Shafer model or hybrid DSm model. Because the extended nine original zones are not exclusive completely in the overlapped model, it is not a Dempster-Shafer model any more. In fact the overlapped model is a hybrid DSm model, not a free DSm model, because there are some exclusivity among some zones but not full non-exclusivity among all zones. For example, based on Figure 70 zone one has intersection with zones 2, 4 and 5 while it is exclusive from zones 3,6,7,8 and 9. Comfort zone, zone 5, is the only one that has intersection with all other zones.

Uncertainty and imperfection force expansion of hypotheses boundaries to adjacent ones and intersect with each other. Although it is feasible in our proposed overlapped model. But in most of the information fusion applications it is so difficult if not impossible to dig further for more granularities in defining elements of FOD. (Because of imprecise, relative, vague and fuzziness of the problem, Small/tall and hot/cold that there is not generally well accepted reference point).

If we define each new decomposed refined area in Figure 70 as a new zones, total 25 zones without any intersection, then we can have Dempster-Shafer model with 25 exclusive and exhaustive hypotheses. In this case, Dempster-Shafer combination rule can be applied to calculate total mass functions on the new 25 hypotheses instead of original 9 hypotheses. Note that we only use this as a ground truth in this study. The number of decomposed area without any intersection with each other will grow exponentially when uncertainties exist, thus in reality it would prevent the use of DST with exclusive and exhaustive hypotheses due to the huge number of the decomposed area. On the contrary, the number of the areas remains the same for DSm hybrid model, as explained later. Table 12 shows the temperature and humidity reference values and intersections for related zones based on the overlapped model.

Simulation and real data analysis of overlapped model:

Similar to Figure 68 for non-overlapped model, to validate second proposed model in Figure 70 we applied synthetic data. Figure 71 shows the results for random test data set based on 25 hypotheses DST model. Decision results based on belief, plausibility and pignistic probability are similar and outperform the decision based on the mass function. Figure 72 shows the total mass, belief functions and related decision making result for our ten hours real data.

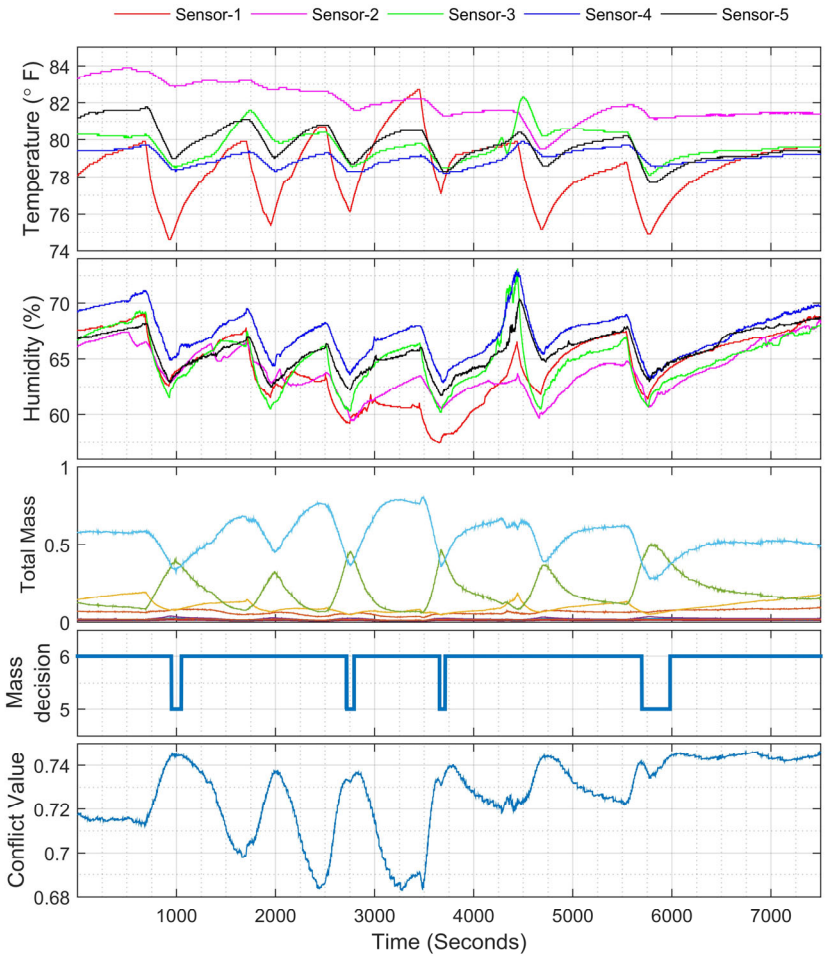


Figure 69: Total mass, decision and conflict for real data (9 hypotheses DST model)

Table 12: Emerged new zones based on proposed 25 hypotheses DST model for summer season

Zone No.	10 18	11 19	12 20	13 21	14 22	15 23	16 24	17 25
Intersection zones	1,2 3,6	2,3 4,7	4,5 5,8	5,6 6,9	7,8 1,2,4,	8,9 2,3,5,	1,4 4,5,7,	2,5 5,6,8,
Temperature ($°F$)	73.4 81.5	78.8 70.7	73.4 76.1	78.8 81.5	73.4 73.4	78.8 78.8	70.7 73.4	76.1 78.8
Humidity (%)	90 70	90 30	50 30	50 30	10 70	10 70	70 30	70 30

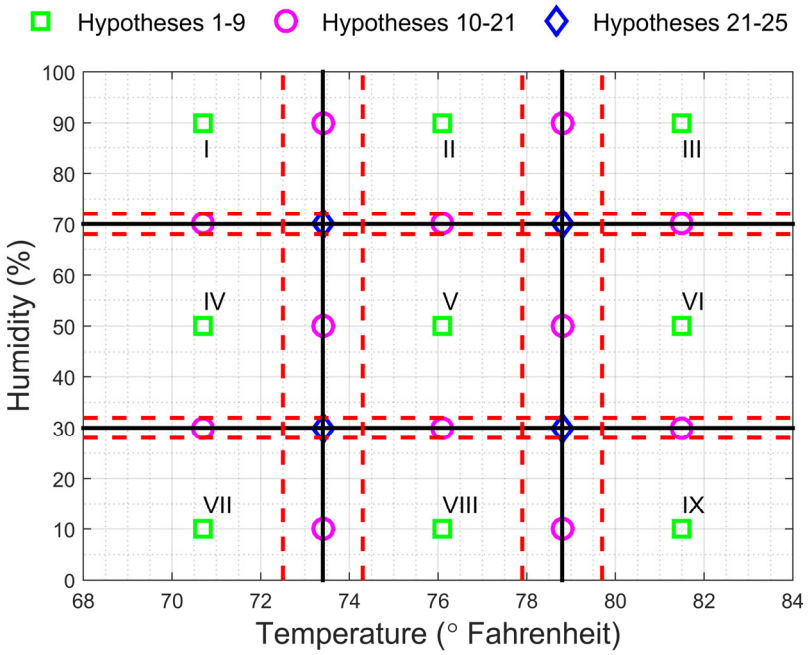


Figure 70: Proposed 25 hypotheses model for summer season (Overlapped model)

According to Figure 72, maximum mass functions move between hypotheses 18 (Intersection between zones 3 and 6 < 36 >, based on Smarandach codification ([301])) and 23 (Intersection among zones 2, 3, 5 and 6 < 2356 >, ([301])). Even if we consider just maximum mass among original focal hypotheses one to nine, it is clear that maximum mass move four times between zone 6 and comfort zone. As a result, decision making by total mass functions do not give reasonable result. It seems belief, plausibility and pignistic probability functions are better for decision making. It is observed in Figure 72 that when AC turned on six times, decision result based on belief (Similar with plausibility and pignistic probability decision) six times transfer to comfort zone (zone 5) from zone 6. Hence the decision making by belief, plausibility and pignistic probability in this proposed overlapped model outperform the non-overlapped model. Nevertheless, Figure 72 shows that conflict values did not decrease for the new model in DST mode and conflict values are even higher.

As an alternative method, we treat Figure 70 as a DSm hybrid model with nine hypotheses. They are not completely exclusive among all hypotheses but they are exhaustive. We applied PCR5 rule based on the quasi method outlined in Section 333. The result in Figure 73 and Figure 74 shows that PCR5 decision is very accurate even if there are only nine hypotheses

in the DSm hybrid model. So although for general model in DST, size of power set for 25 hypotheses is $|\theta| = 2^{25}$, and in free DSmT model cardinality of hyper power set with 9 hypotheses is majored by 2^{29} but with considering the known exclusivity constraints among hypotheses in our proposed models, classic DSmT can be decrease to hybrid DSmT. As a result combination of DST and DSmT-PCR5 for multi-sensor with higher order multi hypotheses can be feasible with lower computation complexity.

Table 13 compares the average run time on the real data for the three related cases. DSmT model with PCR5 needs more computation time in this test case. However, DSmT model will sustain because the number of hypotheses remains the same while DST model will not due to the exponential growth in the number of hypotheses, as explained before.

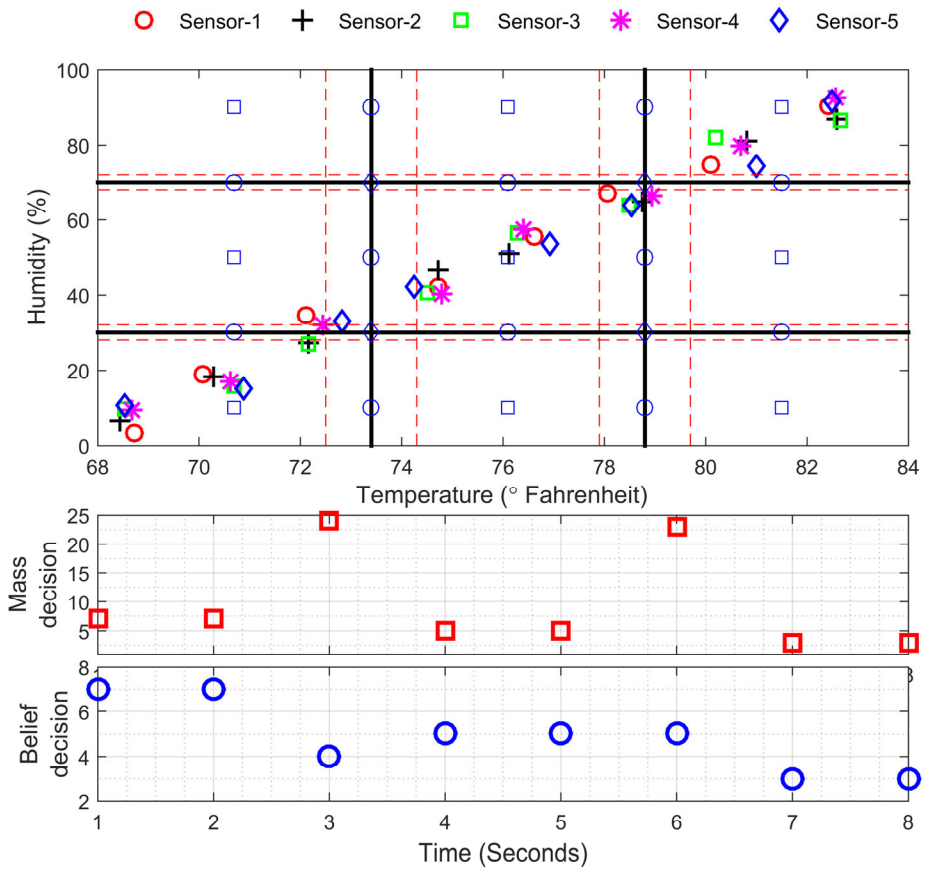


Figure 71: Mass and belief decisions for test data set (25 hypotheses DST model)

Thus it is expected that DSmT model with PCR5 would be appropriate for big data processing with large number of hypotheses or high cardinality.

Furthermore, DSmT model with PCR5 outperforms DST model with the same number of classes by a big margin. Define PD as the detection probability, i.e., correct detection of comfort zone ($P_D = Pr(H_1|H_1)$). Similarly P_F is define as the probability of wrong decision ($P_F = P_r(H_1|H_0)$). Using DST 25 hypotheses model as a ground truth, wecalculate P_D and P_F for DST model (Non-overlapped model) and DSmT model (Overlapped model) both with 9 hypotheses. Table14 shows that DSmT model has much higher P_D = 99.56% comparing to DST 9 model P_D = 36.24%.

Table 13: Average Run-Time on real data for three cases

	DST9	DST25	DSmT9	DSmT4
Avg Run-Time (Seconds)	0.26	0.88	15.82	3.17

Table 14: Probability of Detection and False Alarm Rate

Model	P_D	P_F
DST9 (Non-overlapped model)	36.24%	0
DSmT9-PCR5 (Overlapped model)	99.56%	15.92%
DSmT4-PCR5 (dynamic FOD)	70.66%	0

4.3.1.3 Dynamic FOD Generation

Although applications of DST and DSmT theories are traditionally limited to small number of evidences with few hypotheses, in the previous section we showed that considering known exclusivity among hypotheses in the proposed models could improve the computation efficiency. This opens the door to extend applications of DSmT for multi-sensor with high number of hypotheses. In this section we will consider another method to improve the computational efficiency. The goal is to decrease the cardinality of hypotheses. Considering rational and historical knowledge about the problem, number of hypotheses can be decreased dynamically by incorporating the knowledge. This content based and temporal knowledge can shrink the size of the model and filter some irrelevant evidences.

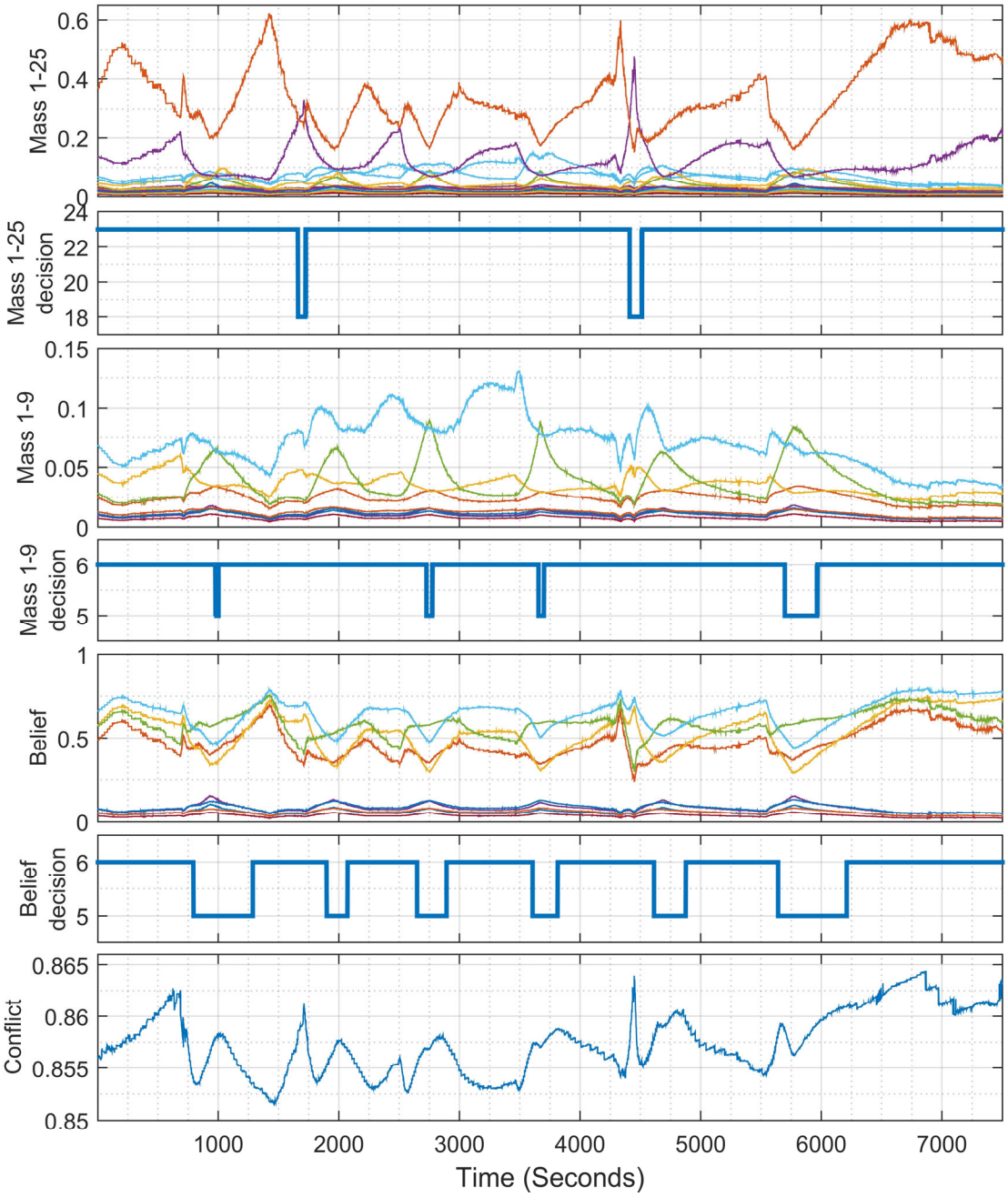


Figure 72: Decision result based on mass and belief for real data (25 hypotheses DST model)

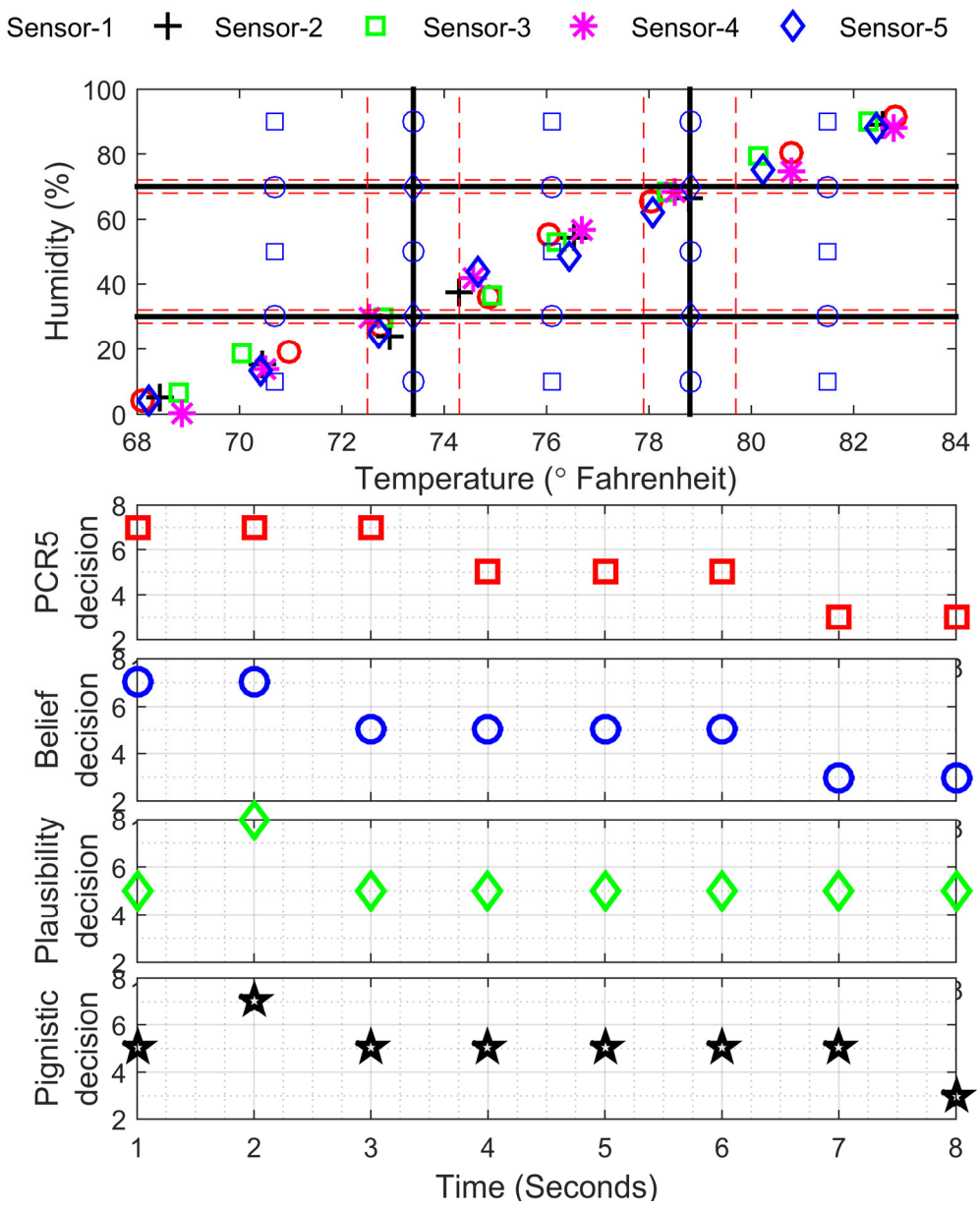


Figure 73: PCR5 decision based for test data (9 hypotheses DSm model)

With prior knowledge about some of the non-existential singletons that not occur surely at certain times and ignoring them, ([302]) showed that the current size of FOD can be decreased dynamically along the time and computational complexity improve exponentially. For example, for our comfort zone case study, we know temperature and humidity are both high in Texas during the summer and vary among zones 2, 3, 5, and 6. Then unrelated data and hypotheses can be filtered as outlier. Similarly for other applications prior knowledge and historical information of the related problem can apply to decrease the size of FOD. If prior information does not available or difficult to access, it is possible to minimize the number of most related hypotheses intelligently. Based on the algorithm shown in Algorithm 8, dynamically related hypotheses could be generated at each time based on the data range.

Here we perform simulation on new FOD based on zones 2, 3, 5, and 6. Figure 75 shows the related temperature and humidity data based on new FOD and their corresponding DSmT-PCR5 combination, belief and associated decision. Based on Figure 75 it detected five transitions out of six from zone 6 into comfort zone. Using DST 25 hypotheses model as a ground truth, we calculate P_D = 70.66% and P_F = 0% for the 4 hypotheses DSmT model. It is clear that comparing to DST 9 hypotheses, P_D improved (doubled) although it is not as accurate as the DSmT 9 hypotheses. Similarly, considering new FOD in DST model we will have nine zones including zones 2, 3, 5, 6 and their intersections (< 23 >, < 25 >, < 36 >, < 56 >, < 2356 >). Figure 76 shows the corresponding DST combination, belief and associated decision. Based on Figure 76 it detected four transitions out of six from zone 6 into comfort zone. So it is not accurate as PCR5 with four hypotheses.

4.3.1.4 Computational Complexity Analysis

In this section we derive the computational complexity of the proposed methods. The general $m \times m$ model for 2 dimension is shown in Figure 77. Assume each zone only has intersection with its direct adjacent neighbors. If we consider it in DSmT platform, the size of FOD (the number of hypotheses) is $|\Theta_{PCR5}| = m^2$. Assuming refinement is accessible, then the size of FOD for DST framework will be $|\Theta_{DST}| = (2m-1)^2$. Because according to Figure 77 in each dimension $(m-1)$ intersection emerged that need to be consider as new refinement zone. If we extend the model to higher dimensions d, related sizes of FOD will be $|\Theta_{PCR5}| = m^d$ and $|\Theta_{DST}| = (2m-1)^d$, respectively for DSmT and DST. So with increasing m, the number of hypotheses

grow exponentially for both DST and DSmT model. The number of power set is approximately 4 times higher than the hyper power set for the same value of m.

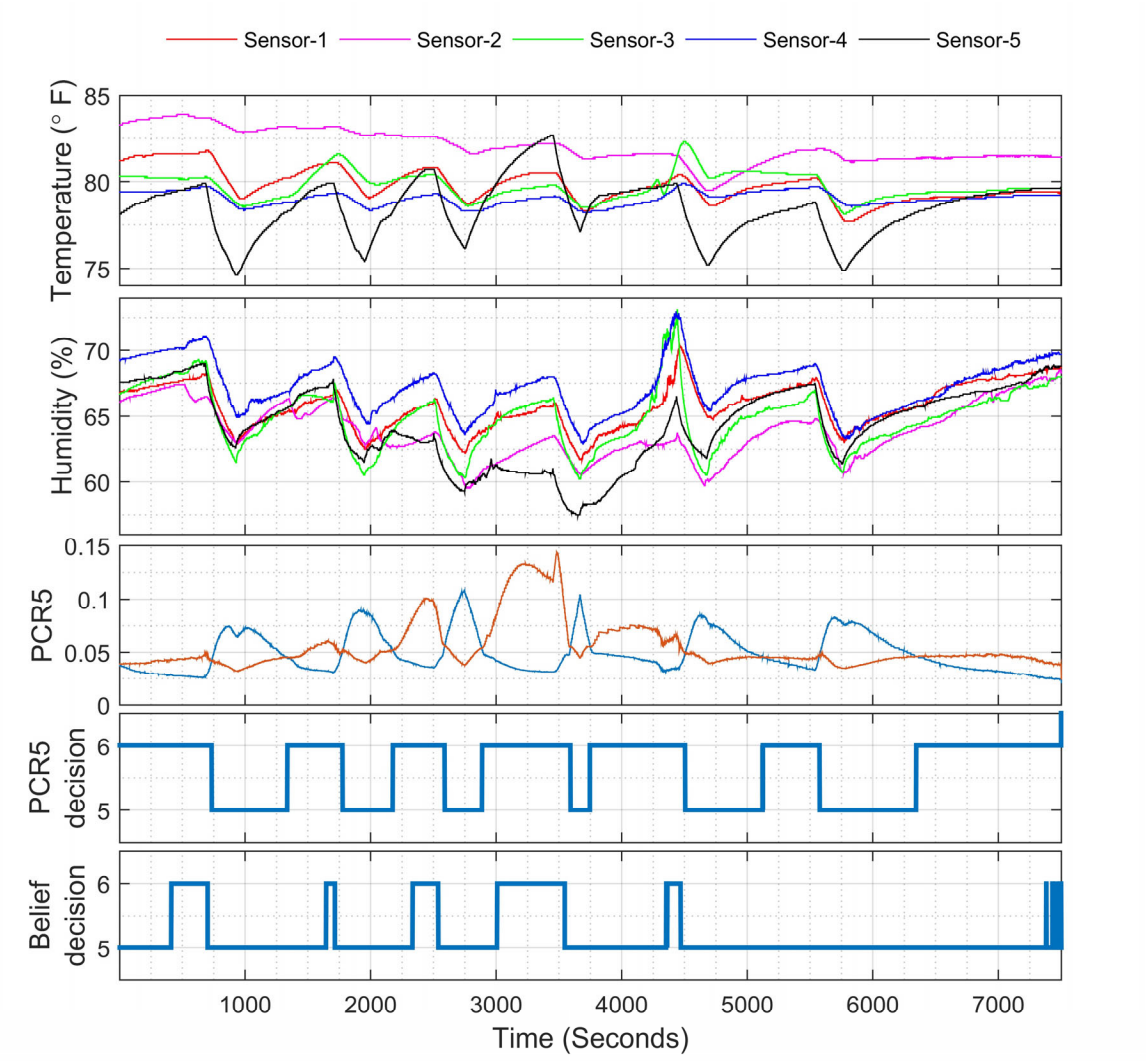


Figure 74: PCR5 and belief decision for real data (9 hypotheses DSm model)

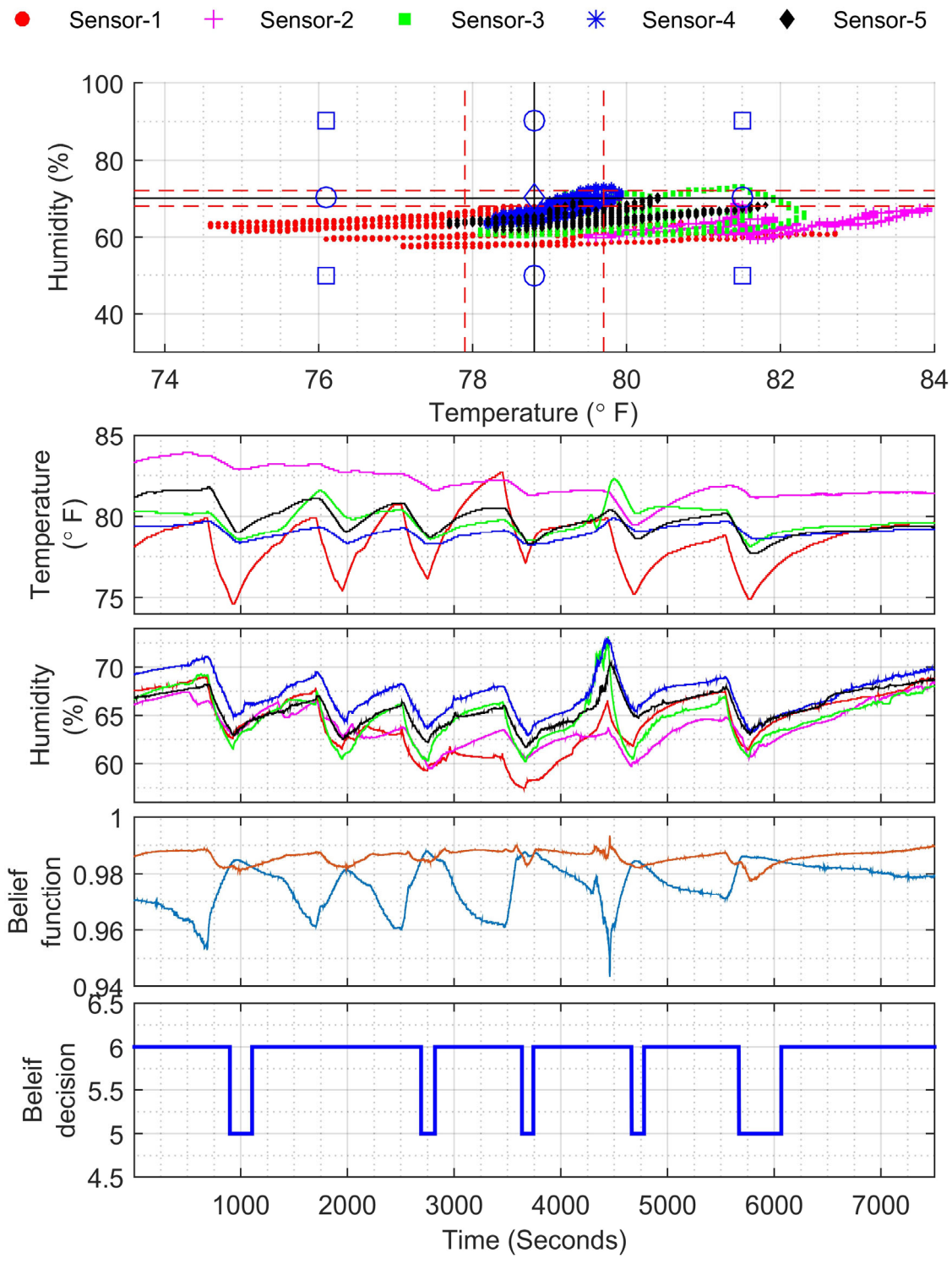


Figure 75: PCR5 and belief decision for real data (4 hypotheses DSm model)

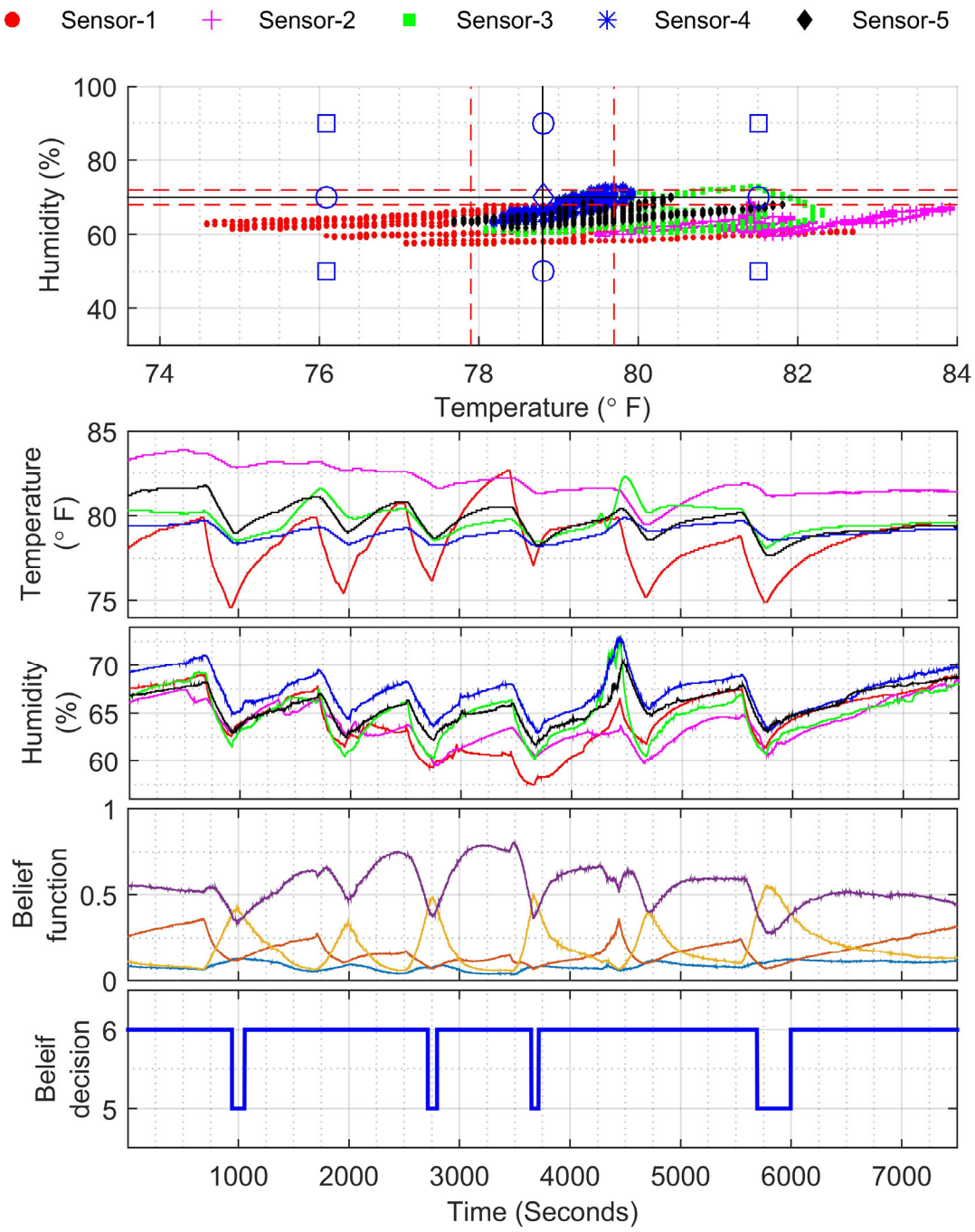


Figure 76: DST and belief decision for real data (9 hypotheses DST model)

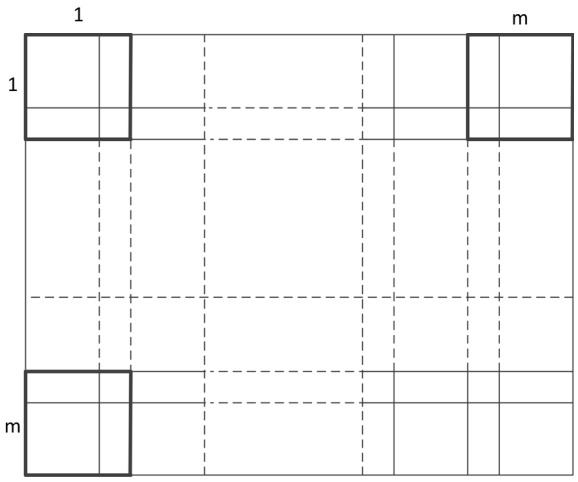


Figure 77: Generalized model size of m×m

Denote the computation of one arithmetical operations: addition, subtraction, multiplication, and division as A, S, M, and D, respectively. The most time consuming operation is division, followed by multiplication and then addition and subtraction, with the last two usually considered together. According to Figure 77 for DST model with n sensors/evidences and cardinality of FOD $|\Theta_{DST}| = (2m-1)^2$, the computation complexity of DST combination rule can be calculated as:

$$o_{\{DST\}}(m,n) = \begin{bmatrix} (3(2m-1)^2+1)M + 3(2m-1)^2A \\ +((2m-1)^2+1)D + S \end{bmatrix}(n-1) \tag{4.20}$$

$$o_{\{DST\}}(m,n) \approx m^2 n \tag{4.21}$$

Similarly for DSmT, PCR5 combination rule among n sensors/evidences and cardinality of FOD $|\Theta_{PCR5}| = m^2$, in the way we applied to keep commutativity and associativity, the computational complexity is given by:

$$o_{\{PCR5\}}(m,n) = [(3M+2A)m^2 + (5M+4A)2m(m-1)$$
$$+(17M+16A)(m-1)^2 + M](n-1) + (2M+A+D)$$
$$[(m^2+1)[(2m-1)^2+1] - 3m^2 - 10m(m-1)$$

$$-17(m-1)^2-1] \tag{4.22}$$

$$o_{\{PCR5\}}(m,n) \approx m^2 n + m^4 \tag{4.23}$$

Here the first term is related to $n-1$ times conjunctive consensus among the n sensors. The second term represents the computation need to redistribute partial conicts. m^2 is the cardinality of FOD or the number of singleton focal elements. $2m(m-1)$ is the number of intersections between two adjacent hypotheses. $(m-1)^2$ is the number of intersections among four neighbor hypotheses.

Algorithm 8 Dynamic frame of discernment (FOD) generation & combination

Input: Time series data from "*N*" sensors
Output: FOD generation, DSmT combination, and decision
for *t* = 1 : *t time* **do**
  for *i* = 1 : *N* **do**
    Find the maximum(*max*) & *minimum*(*min*)
  end for
  Generate FOD [*m : n*] *based on max & min*
  Calculate distance functions
  Calculate mass functions (bba)
  Compute DSmT − PCR5 combination
  Compute belief (*bel*)
  Compute plausibility (*pl*)
  Compute pignistic probability (pp)
  If $pp(i) \geq pp(j), i, \forall j \in [m{:}\, n], j \neq i$" **then**
    Declare "i" as decision making hypothesis
  end if
end if

Table 15 provides some examples of the computation complexity for our proposed models in the previous sections. Although the cardinality of DST is higher than DSmT, the computational complexity of DSmT is higher based on Equations 4.21 and 4.23. Contrary to DST, in DSmT-PCR5 model during PCR5 combination, new focal elements are generated and all exclusive intersections as a partial conflict redistributed. Furthermore, in DST model just conjunctive consensus is calculated among the focal elements. However, due to the exponential growth in the number of classes

in DST model, it may be difficult to keep track all of them and some classes may not be easily characterized because of the uncertainty. Furthermore, computation complexity to calculate belief functions, plausibility and pignistic probability for both DST and DSmT models can be given by following. For DST model because of specific structures of our proposed models, the computation complexity for those three metrics are approximately similar. Then here we only show the computation complexity for belief functions. If belief function is needed after mass combination calculated between two sources of evidences then computation complexity will be:

$$o_{\{BelDST\}}(m,n) = [8(2m-3)^2 + 20(2m-3) + 12]A(n-1)$$
$$\approx 32m^2nA \approx m^2n \tag{4.24}$$

While if belief function applied to the final step on the total mass function, so regardless of the number of sensors/evidences computation complexity can be shown by:

$$o_{\{BelDST\}}(m,n) = [8(2m-3)^2 + 20(2m-3) + 12]A$$
$$\approx 32m^2 \approx m^2 \tag{4.25}$$

Here, 12 is the number of addition needed to calculate belief functions related to the four corner hypotheses, three for each one. $20(2m-3)$ is for $4(2m-3)$ hypotheses between corners in the periphery, five for each one. And finally, $8(2m-3)^2$, is for $(2m-3)^2$ inner hypotheses that each one with eight addition.

For DSmT model, computation complexity to calculate belief functions after mass combination calculated between two sources of evidences will be similar to DST model only with different in the number of hypotheses and can be shown by:

$$o_{\{BelDSmT\}}(m,n) = [8(m-2)^2 + 20(m-2) + 12]A(n-1)$$
$$\approx 8m^2n \approx m^2n \tag{4.26}$$

Similarly, for DSmT model if belief functions applied on the total mass function, so regardless of the number of sensors/evidences computation complexity can be shown by:

$$o_{\{BelDSmT\}}(m) = [8(m-2)^2 + 20(m-2) + 12]A \approx 8m^2 \approx m^2 \quad (4.27)$$

Here, $(m-2)$ is the number of hypotheses between two hypotheses in the corner and $(m-2)$ is the total number of inner hypotheses. For DSmT model computation complexity of plausibility and pignistic probability is different with DST one and can be calculated for both situations respectively:

$$o_{\{PlDSmT\}}(m,n) = 2[8(m-2)^2 + 20(m-2) + 12]A(n-1) \approx 16m^2n \approx m^2n \quad (4.28)$$

$$o_{\{PlDSmT\}}(m) = 2[8(m-2)^2 + 20(m-2) + 12]A \approx 16m^2 \approx m^2 \quad (4.29)$$

Table 16 provides some examples of the computation complexity of belief function, plausibility and pignistic probability for our proposed models. It is clear that to calculate belief function and plausibility DSmT model need lower computation, at least two times, compare to DST one.

Table 15: Computation Complexity of mass function (n=5)

Operation	m	# M	# A	# D	# S
DST-25 hypotheses	3	304	300	104	4
DST-9 hypotheses	2	112	108	40	4
DSmT-9 hypotheses	3	832	624	104	0
DSmT-4 hypotheses	2	200	160	0	0

Table 16: Computation Complexity of Belief and Plausibility/Pignistic Probabilities (n=5)

Operation	# A(m=3)	# A(m=4)	# A(m=10)
DST-Bel/Pl/betP	576	1248	10656
DSmT-Bel	160	336	2736
DSmT-Pl/betP	320	672	5472

4.3.2 Semi-supervised Learning

4.3.2.1 Datasets

To validate the effectiveness of the proposed framework on fake news detection, we test the implemented model on two benchmarks: LIAR and PHEME.

LIAR: LIAR [208] is the recent benchmark dataset for fake news detection. This dataset includes 12,836 real-world short statements collected from PolitiFact[10], where editors handpicked the claims from a variety of occasions such as debate, campaign, Facebook, Twitter, interviews, ads, etc. Each statement is labeled with six-grade truthfulness, namely, true, false, half-true, part-fire, barely-true, and mostly-true. The information about the subjects, party, context, and speakers are also included in this dataset. In this paper, this benchmark contains three parts: training dataset with 10,269 statements, validation dataset with 1,284 statements, and testing dataset with 1,283 statements. Furthermore, we reorganize the data as two classes by treating five classes including false, half-true, part-fire, barely-true, and mostly-true as Fake class and true as True class. Therefore, the fake news detection on this benchmark becomes a binary classification task.

PHEME: We employ PHEME [218] to verify the effectiveness of the proposed framework on social media data, where it collects 330 twitter threads. Tweets in PHEME are labeled as true or false in terms of thread structures and follow-follower relationships. PHEME dataset is related to nine events whereas this paper only focuses on the five main events, namely, Germanwings-crash (GC), Charlie Hebdo (CH), Sydney siege (SS), Ferguson (FE), and Ottawa shooting (OS). It has different levels of annotations such as thread level and tweet level. We only adopt the annotations on the tweet level and thus classify the tweets as fake or true. The detailed distribution of tweets and classes is shown in Table 18 after the data is preprocessed such as removing data redundancy. It is observed that the class distribution is different among these events. For example, three events including CH, SS, and FE have more true news while the event GC and OS have more fake news. Furthermore, class distributions for events such as CH, SS, FE are significantly imbalanced, which

[10] https://www.politifact.com

will be a challenge to the detection task (a binary classification task). In addition, the word distributions vary among five events, where one example is shown in Figure 78: there are 50 words ranked by their TF-IDF values, where TFIDF values are used to evaluate the word relevance to the event [303]. It is observed that the top ranked words are very different for different events. Moreover, even for the same word, their relevance are not the same for different events. For example, the relevance (TF-IDF values) of the word "*thank*" are different between events including

Charlie Hebdo (CH), Ferguson (FE), and Ottawa shooting (OS). In addition, we perform leave-one-event-out (LOEO) cross-validation [304], which is closer to the realistic scenario where we have to verify unseen truth. For example, the training data can contain the events GC, CH, SS, and FE whereas the testing data will contain the event OS. However, as highlighted in Figure 78, this fake news detection task becomes very difficult since the training data has very different data distribution from that of the testing data.

It should be noted that *the data in original datasets is fully labeled. We employ all labeled data to train the baseline models*. Compared to the baselines, *we train the proposed models with only partially labeled data and the percentage of labeled data* for training of our proposed models is from 1% to 30%.

4.3.2.1 Experiment Setup

The key hyper-parameters for the implemented model are shown in Table 17 and we employ Adam optimizer to complete the training the implemented proposed model (TDSL).

Table 17: Hyper-parameters for the training of Word CNN based TDSL

Hyper-parameters	Values
Dropout	0.5
Minibatch size	128
Number of epochs	200
Maximum learning rate	1e-4

In addition, we employ five deep supervised learning models as baselines including 1) Word-level CNN (Word CNN) [217], 2) Character-level CNN (Char CNN) [305], 3) Very Deep CNN (VD CNN) [306], 4) Attention-Based Bidirectional RNN (Att RNN) [307], and 5) Recurrent CNN (RCNN) [308], where these models perform well on text classification. Specifically, Word CNN

performs well on sentence classification, which is more suitable to process social media data as the length of the content of the data is short like that of the sentence. In addition, we build 6) word-level bidirectional RNN (Word RNN) to compare the implemented model, where Word RNN contains one embedding layer and one bidirectional RNN layer, and concatenate all the outputs from the RNN layer to feed to the final layer that is a fully-connected layer. Thus, there are total 6 baseline models. *Note that baseline models use all labeled data from the original datasets*. Moreover, we apply label propagation [183] as a baseline to compare our model from semi-supervised learning point of view as it has been employed for text classification such as Twitter polarity classification [185] and fake news detection [186].

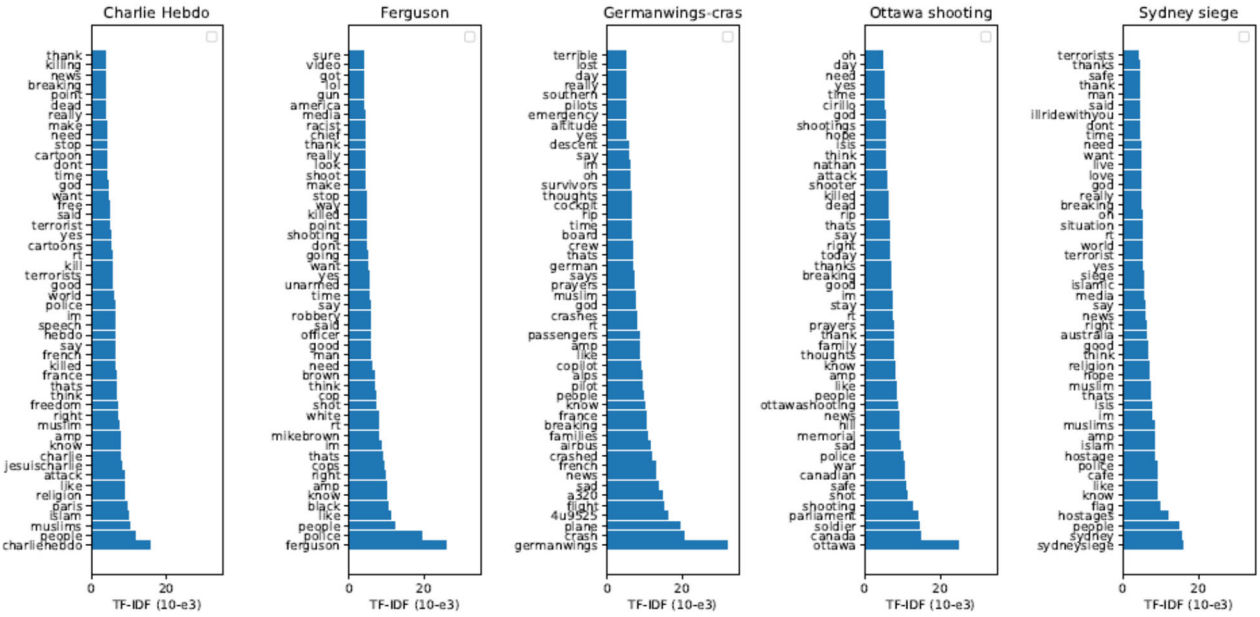


Figure 78: An example of the difference of word distributions between five events in PHEME dataset. x-axis indicates the TF-IDF value of the word while y-axis shows top 50 words ranked by corresponding TF-IDF values.

Table 18: Number of tweets and class distribution in the PHEME dataset.

Events	**Tweets**	**Fake**	**True**
Germanwings-crash	3,920	2,220	1,700
Charlie Hebdo	34,236	6,452	27,784
Sydney siege	21.837	7,765	14,072
Ferguson	21,658	5,952	15,706
Ottawa shooting	10,848		
Total	**92,499**	**27,992**	**64,507**

4.3.2.2 Evaluation

We apply different metrics to evaluate the performance of fake news detection regarding the task features on these two benchmarks.

- **LIAR**: We employ accuracy, precision, recall and Fscore to evaluate the detection performance. Accuracy is calculated by dividing the number of statements detected correctly over the total number of statements.

$$Accuracy = \frac{N_{correct}}{N_{total}}. \tag{4.30}$$

In addition, we employ Fscore values of each class to check the performance since the task is a binary text classification.

$$Fscore = \frac{2 \times Precision \times Recall}{Precision + Recall} \tag{4.31}$$

where Precision indicates precision measurement that defines the capability of a model to represent only fake statements and Recall computes the aptness to refer all corresponding fake statements:

$$Precision = \frac{TP}{TP + FP} \tag{4.32}$$

$$Recall = \frac{TP}{TP + FN} \tag{4.33}$$

whereas TP (True Positive) counts total number of news matched with the news in the labels. FP (False Positive) measures the number of recognized label does not match the annotated corpus dataset. FN (False Negative) counts the number of news that does not match the predicted label news.

- PHEME: Accuracy is one of the common evaluation metric to measure the performance of fake news detection on this dataset [173]. However, we also evaluate the performance based on the Fscore since our task on PHEME datasets is the binary text classification with imbalanced data. Specifically, as we perform leave-one-event-out cross-validation on the PHEME dataset, we utilize macro-averaged Fscore [309] to evaluate the whole performance of mining fake news on different events.

$$MacroF = \frac{1}{T}\sum_{t=1}^{T} Fscore_t. \tag{4.34}$$

$$MacroP = \frac{1}{T}\sum_{t=1}^{T} Precision_t. \tag{4.35}$$

$$MacroR = \frac{1}{T}\sum_{t=1}^{T} Recall_t. \tag{4.36}$$

where T denotes the total number of events and $Fscore_t$, $Precision_t$, $Recall_t$ are *Fscore*, *Precision*, *Recall* values in the t^{th} event. Additionally, we use macro-average accuracy on five events to examine performance. The main goal for learning from imbalanced datasets is to improve the recall without hurting the precision. However, recall and precision goals can be often conflicting, since when increasing the true positive (TP) for the minority class (True), the number of falsepositives (FP) can also be increased; this will reduce the precision [310]. It means that when the MacroP is increased, the MacroR might be decreased for the case of PHEME.

$$MacroA = \frac{1}{T}\sum_{t=1}^{T} Accuracy_t. \tag{4.37}$$

4.3.2.3 Results and Discussion

We compare the two-path deep semi-supervised learning (TDSL) implemented based on Word CNN with the baselines on two datasets: LIAR and PHEME. In addition, we examine the effects

on the performance when applying different hyper-parameters. All evaluation results are average values on five runs.

LIAR: Table 19 presents the performance comparison on LIAR datasets. When we focus on the baselines, Word CNN outperforms other baselines with respect to the accuracy, recall, and Fscore. However, with respect to Precision, Att RNN is better than other baselines. It means we should select specific models when we concern certain evaluation metrics. In addition, we present results generated by the proposed TDSL, where we utilize 1% and 30% of labeled data and the rest of unlabeled data to build the deep semi-supervised model. It is observed that the performance is strengthened by increasing the amount of the labeled data for training TDSL. Specifically, even we use little amount of labeled data, we still obtain acceptable performance. For example, we use 1% labeled training data to construct Word CNN based TDSL, compared with the Word CNN, its accuracy and Fscore are only reduced about 6% and 3%, respectively. Moreover, the accuracy, recall and Fscore of TDSL (1%) are better than those of some baselines such as Char CNN, RCNN, Word RNN, and Att RNN. It means that the deep semi-supervised learning built based on the proposed framework is able to detect fake news even with little labeled data. Moreover, when we compare our model with label propagation, we observe that our model could obtain higher performance with more labeled data in terms of Fscore values.

In Table 20, we show how the ratio of labeled data affect the detection performance generated with TDSL. When we increase the ratio of labeled data step by step, the performances such as accuracy, recall, and Fscore are improved significantly while only precision is relatively stable. Moreover, when we use the 30% labeled data, the performance is similar to the-state-of-the-art obtained with Word CNN. Specifically, the precision is decreased slightly when increasing the ratio of labeled data. The supervised loss, cross entropy, is defined in terms of the prediction a curacy. Therefore, we are able to gain higher accuracy when adding more labeled data into the training procedure, but there is no guarantee to increase precision.

Additionally, we examine the performance effects from different hyper-parameters in Figures 79, 80, and 81. In Figure 79, we focus on the effects from different batch sizes. When we train the model with fewer labeled data, the different batch sizes affect the performance more significantly. For instance, the performance hardly change when there are 30% labeled data while the performance varies when there are only 1% labeled data. It is because more information on the ground

truth will be embedded in the training samples when batch size is large, which results in robust prediction.

For Figure 80 and 81, we examine the performance effects on different embedding sizes and learning rates. In Figure 80, there are similar observations to those of the case of batch size. For example, for the case of more labeled data for training, different embedding sizes doesn't affect the performance significantly. On the contrary, for the fewer labeled data, choosing embedding size should be carefully because different embedding sizes will lead to different performances. In addition, larger embedding size enhances the performance. In Figure 81, we observe that there is no significant difference when using different learning rates in the case of 10% labeled data and 30% labeled data while only small performance difference can be observed in the case of 1% labeled data.

Table 19: Comparing performance between baselines and proposed model (TDSL) on LIAR Datasets. The baselines, namely, Word CNN, Char CNN, VD CNN, RCNN, WORD RNN, and Att RNN, are built with the training data that is fully labeled. On the contrary, we only apply 1% and 30% labeled training data and rest of unlabeled training data to accomplish learning of the proposed model.

Model	Accuracy	Precision	Recall	Fscore
Word CNN [217]	**85.01%**	83.57%	99.94%	**91.02%**
Char CNN) [305]	77.93%	84.09%	91.41%	87.59%
VD CNN [306]	83.77%	83.56%	99.88%	91.00%
RCNN [308]	79.30%	84.19%	89.66%	86.81%
Word RNN	72.44%	84.19%	81.52%	82.79%
Att RNN [307]	75.90%	84.26%	86.79%	85.52%
Label Propagation (1%)	83.16%	83.48%	99.53%	90.80%
Label Propagation (30%)	81.21%	83.31%	96.92%	89.60%
TDSL (1%)	**79.81%**	**83.62%**	**94.30%**	**88.62%**
TDSL (30%)	**83.36%**	**83.59%**	**99.64%**	**90.91%**

Table 20: Comparing performances generated by proposed model (TDSL) learning on different ratios of labeled training data and rest of unlabeled training data.

Ratio of Labeled	Accu-	Preci-	Re-	Fscore
1%	79.81%	83.62%	94.30	88.62%
3%	80.71%	83.69%	95.54	89.21%
5%	80.74%	83.70%	95.59	89.23%
8%	82.24%	83.56%	98.02	90.21%
10%	82.52%	83.58%	98.41	90.39%
30%	83.36%	83.59%	99.84	90.91%

Table 21: Comparing performance between baselines and proposed model (TDSL) on PHEME Datasets. The baselines, namely, Word CNN, Char CNN, VD CNN, RCNN, WORD RNN, and Att RNN, are built with the training data that is fully labeled. On the contrary, we only apply 1% and 30% labeled training data and rest of unlabeled training data to accomplish learning of the proposed model.

Model	MacroA	MacroP	MacroR	MacroF
Word CNN [217]	61.75%	**50.82%**	17.60%	24.03%
Char CNN) [305]	63.68%	50.66%	19.91%	26.73%
VD CNN [306]	**65.42%**	49.21%	**30.04%**	**28.50%**
RCNN [308]	60.62%	45.86%	16.40%	22.24%
Word RNN	59.70%	45.57%	22.89%	28.22%
Att RNN [307]	60.32%	45.58%	25.49%	31.15%
Label Propagation (1%)	64.19%	38.27%	1.61%	3.08%
Label Propagation (30%)	64.16%	40.59%	2.51%	4.68%
TDSL (1%)	**56.19%**	**38.83%**	**18.73%**	**21.13%**
TDSL (30%)	**60.64%**	**41.14%**	**4.77%**	**6.75%**

Table 22: Comparing performances generated by proposed model (TDSL) learning on different ratios of labeled training data from PHEME Datasets.

Labeled Ra-	Macro	Macro	Macro	Macro
1%	56.19%	38.83%	18.73%	21.13%
3%	58.58%	39.38%	13.12%	17.83%
5%	58.40%	39.18%	12.58%	16.31%
8%	59.74%	40.48%	8.11%	11.18%
10%	59.84%	40.38%	7.08%	10.49%
30%	60.64%	41.14%	4.77%	6.75%

Table 23: Comparing performance with different batch sizes on PHEME Datasets. We choose three cases of ratios of labeled training data, namely, 1%, 10%, and 30%.

	1% Labeled Data			
Batch size	**MacroA**	**MacroP**	**MacroR**	**MacroF**
128	56.19%	38.83%	18.73%	21.13%
256	57.78%	39.72%	18.50%	23.52%
512	57.78%	39.07%	15.73%	20.43%
	10%Labeled Data			
Batch size	**MacroA**	**MacroP**	**MacroR**	**MacroF**
128	59.84%	40.38%	7.08%	10.49%

256	60.08%	40.46%	8.55%	12.49%
512	59.02%	40.81%	12.96%	17.18%
	30%Labeled Data			
Batch size	**MacroA**	**MacroP**	**MacroR**	**MacroF**
128	60.64%	41.14%	4.77%	6.75%
256	60.59%	41.50%	7.09%	10.77%
512	59.94%	42.77%	8.51%	12.27%

Table 24: Comparing performance with different embedding sizes on PHEME Datasets. We choose three cases of ratios of labeled training data, namely, 1%, 10%, and 30%.

	1% Labeled Data			
Embedding	**Macro**	**MacroP**	**Macro**	**Macro**
64	57.38%	39.20%	16.57%	20.78%
128	56.19%	38.83%	18.73%	21.13%
256	57.13%	38.86%	18.07%	22.23%
	10% Labeled			
Embedding	**Macro**	**MacroP**	**Macro**	**Macro**
64	59.73%	40.44%	7.91%	11.06%
128	59.84%	40.38%	7.08%	10.49%
256	58.89%	40.27%	11.67%	15.22%
	30% Labeled			
Embedding	**Macro**	**MacroP**	**Macro**	**Macro**
64	60.79%	42.32%	4.37%	6.99%
128	60.64%	41.14%	4.77%	6.75%
256	60.63%	42.54%	4.63%	6.66%

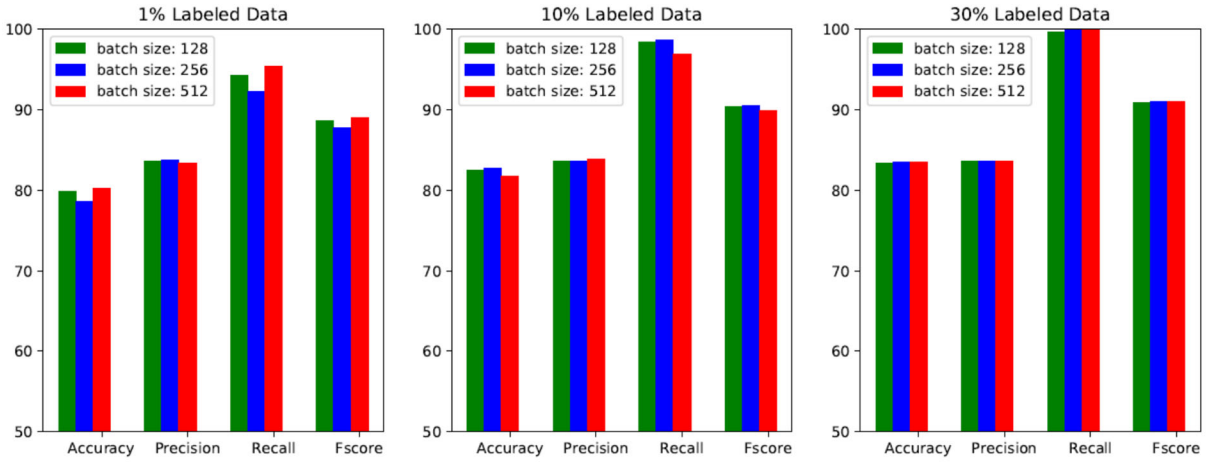


Figure 79: Different performances generated with three batch sizes, 128, 256, and 512 on three ratios of labeled data, namely 1%, 10%, and 30%. x-axis is for different evaluation metrics while y-axis is for performance. Different color bars illustrate different batch sizes, where green bars are for batch size 128, blue bars are for batch size 256, and red bars are for batch size 512.

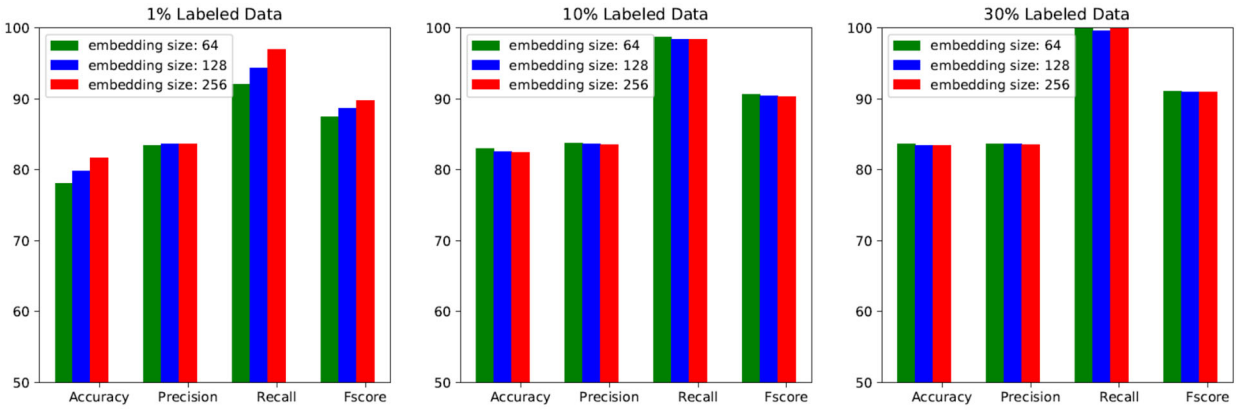


Figure 80: Different performances generated with three embedding sizes, 64,128, and 256 on three ratios of labeled data, namely 1%, 10%, and 30%. x-axis is for different evaluation metrics while y-axis is for performance. Different color bars show different batch sizes, where green bars are for embedding size 64, blue bars are for embedding size 128, and red bars are for embedding size 256.

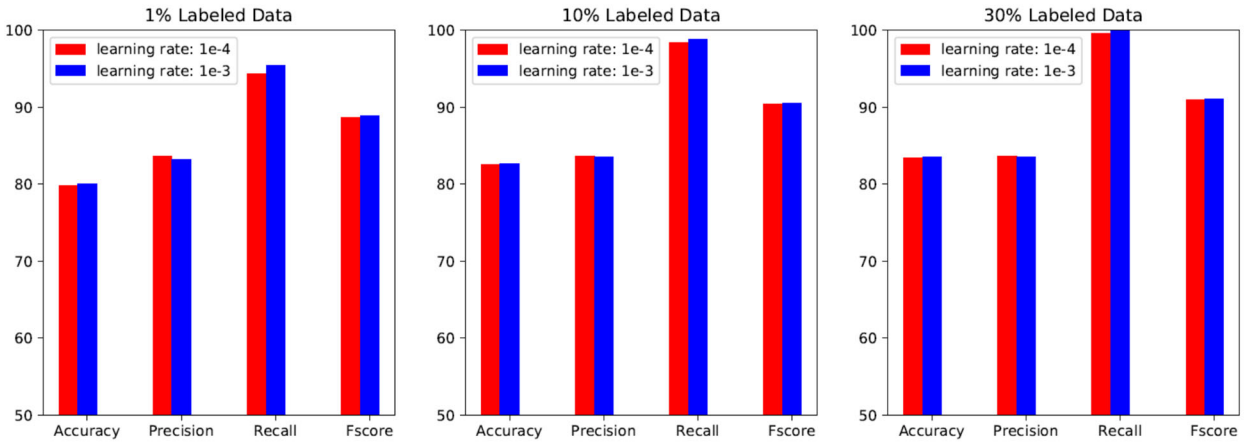


Figure 81: Different performances generated with three learning rate, 1e-3 and 1e-4 on three ratios of labeled data, namely 1%, 10%, and 30%. x-axis is for different evaluation metrics while y-axis is for performance. Different color bars indicate different batch sizes, where blue bars are for learning rate 1e-3, and red bars are for learning rate 1e-4.

4.3.2.4 PHEME

Compared to LIAR dataset, PHEME datasets will introduce new challenges such as imbalanced class distribution and word distribution differences among these events. Therefore, it will lead to different observations, compared to the case of LIAR. Table 21 indicates the performance comparison on PHEME datasets. When we examine the baselines, VD CNN outperforms other baselines with respect to the MacroA, MacroR, and MacroF. However, considering MacroP, Word CNN is better than other baselines. In addition, we observe that the performance (MacroA) is enhanced when increasing the ratio of the labeled data for training TDSL. Moreover, even we use little amount of labeled data, we still obtain acceptable performance. For example, we use 1% labeled training data to construct Word CNN based TDSL, compared with the VD CNN, its MacroA and MacroF are just reduced about 9% and 7%, respectively. However, the MacroF is decreased when MacroA is increased when adding more labeled data for training. There are two reasons for this observation. One is that the learning of TDSL aims to optimize the accuracy, but not the Fscore. The other is that the data distribution of training data is different from that of testing data since we utilize the leave one-out policy to complete the validation, which breaks the assumption that the training data should share the same distribution to the testing data. The more labeled data is, the more serious the

difference on the distribution is. Specifically, our model outperforms label propagation significantly when examining Fscore values. It means that our model can perform better to process fake news detection when there is the difference of word distributions for the leave-one-out evaluation.

Similar to the case of LIAR, in Table 22, we illustrate how the ratio of labeled data affects the detection performance. When we increase the ratio of labeled data step by step, the MacroA is improved as well, but the MacroF is reduced significantly. Specifically, MacroR and MacroF are decreased significantly when increasing the ratio of labeled data. The supervised loss, cross entropy, is defined in terms of the prediction accuracy. Therefore, there is no guarantee to increase MacroR and MacroF when adding more labeled data into the training procedure.

In Tables 23 and 24, we examine the performance differences when choosing different batch sizes and embedding sizes, respectively. We observe the similar trends regarding the MacroA and MacroP, where there is no big difference on MacroA and MacroP when choosing different batch sizes and embedding sizes for building the proposed model. However, MacroR is changed more significantly by different batch sizes when comparing to the case of embedding sizes.

Moreover, in the Figure 82, 83, and 84, we show the detailed performances for five events when choosing different batch sizes to train the model on different ratios of labeled data. When examining the results shown in Figure 82, MacroA is increased for these events Charlie Hebdo, Sydney siege, and Ferguson when increasing the amount of labeled data whereas for the events Germanwings-cras and Ottawa shooting, the MacroA is decreased. It is because more imbalanced classes involved in the training procedure will lead to reducing the performance. For MacroR and MacroF, the performance is reduced when adding the ratios of labeled data. In addition, the similar observations can be obtained in terms of results shown in Figure 83, and 84. However, the difference is that larger batch sizes can reduce the performance affections that are from batch sizes.

Finally, we examine the performance differences when choosing two different embedding sizes, namely 64 and 256, where the results are shown in Figure 85 and 86, respectively. MacroA and MacroP are increased for the events Charlie Hebdo, Sydney siege, and Ferguson when increasing the amount of labeled data for training models whereas for the events Germanwings-cras and Ottawa shooting, the MacroA is decreased. It is caused by the same reason of the case of batch size.

In summary, in terms of observations that are from the aforementioned results, increasing the amount of labeled data for training TDSL will enhance performance when training data has the

similar distribution to the testing data, for instance, the case of LIAR. In addition, for both of benchmarks, we can obtain acceptable performance even using extremely limited labeled data for training. However, we should pay more attention to choosing the ratio of labeled when processing imbalance classification task, for example, the case of PHEME. Meantime, we should delicately choose the hyperparameters if we plan to obtain reasonable performance.

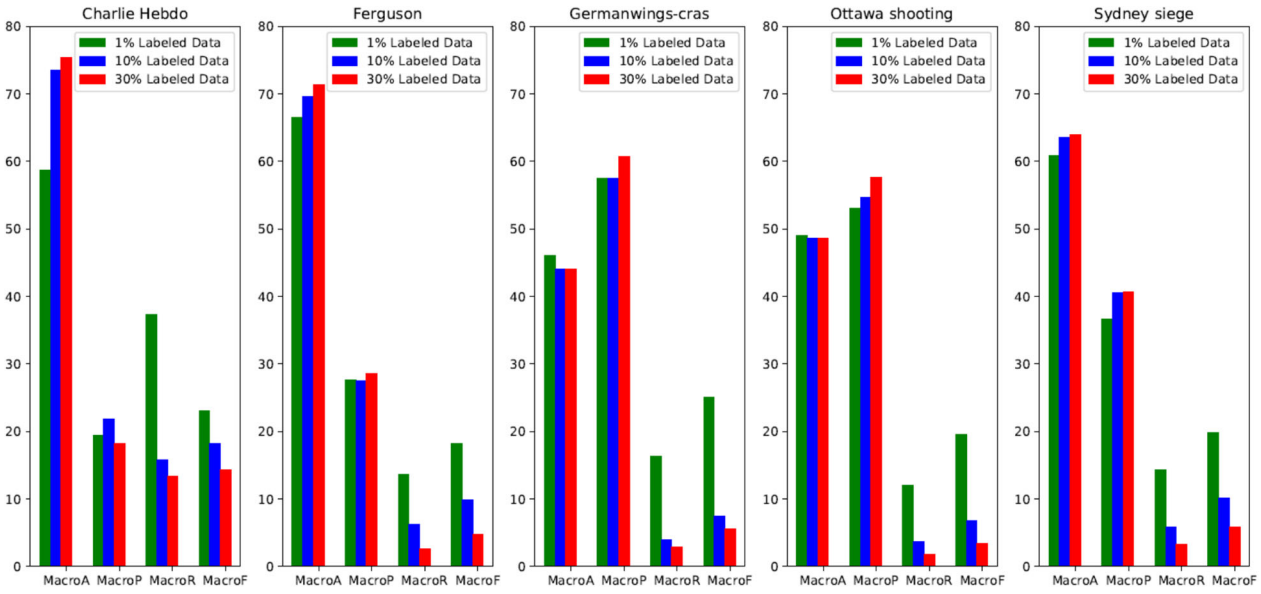


Figure 82: Comparing detailed performances generated with batch size 128 for five events. x-axis is for different evaluation metrics while y-axis is for performance. Different color bars are for different ratios of labeled data, where green bars are for 1%, blue bars are for 10%, and red bars are for 30%

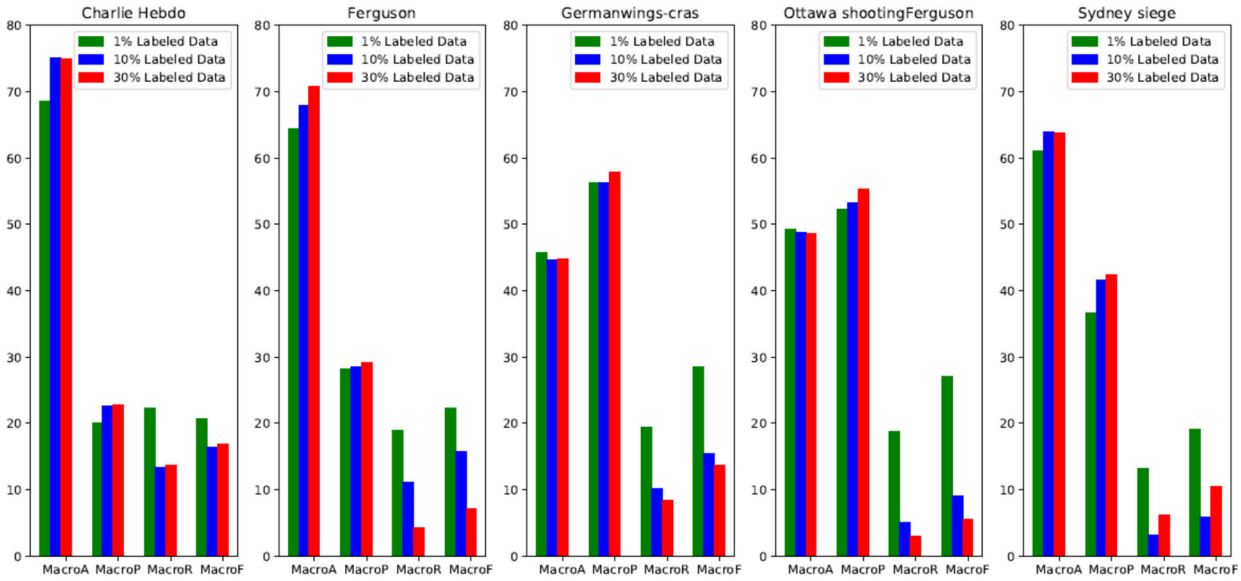


Figure 83: Comparing detailed performances generated with batch size 256 for five events. x-axis is for different evaluation metrics while y-axis is for performance. Different color bars show different ratios of labeled data, where green bars are for 1%, blue bars are for 10%, and red bars are for 30%

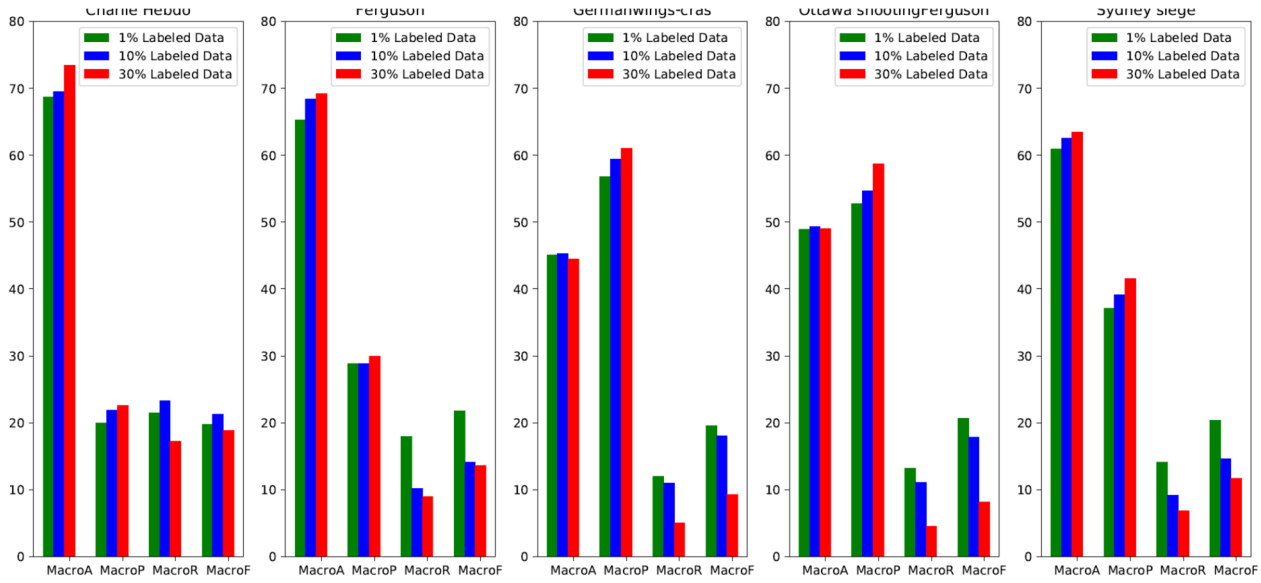


Figure 84: Comparing detailed performances generated with batch size 512 for five events. x-axis is for different evaluation metrics while y-axis is for performance. Different color bars indicate different ratios of labeled data, where green bars are for 1%, blue bars are for 10%, and red bars are for 30%.

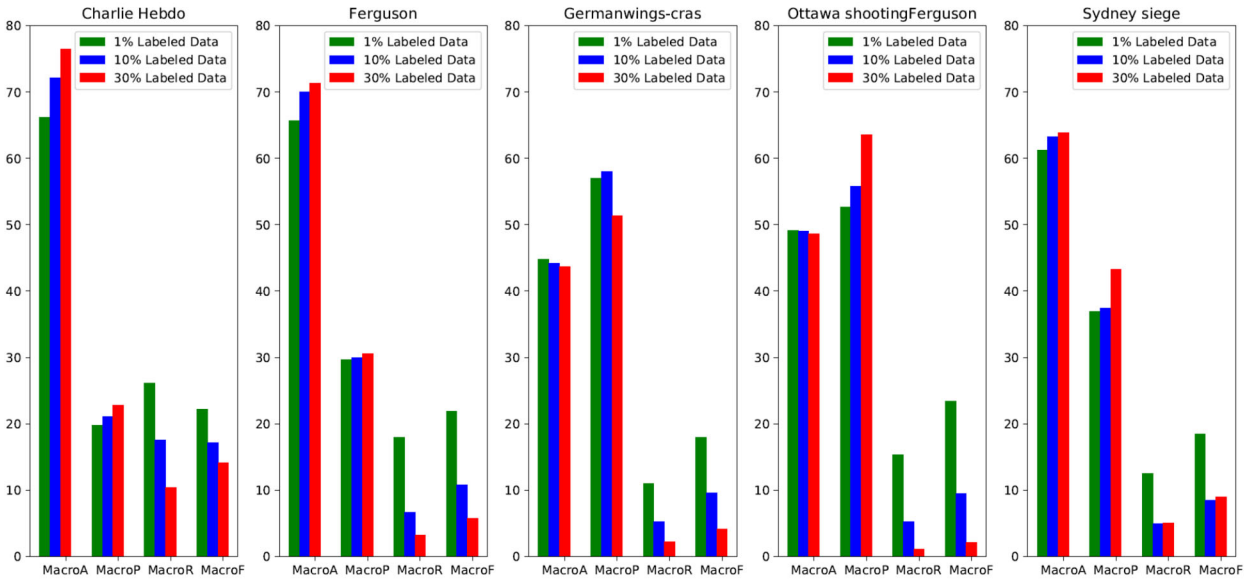


Figure 85: Comparing performance for the case of embedding size 64. x-axis is for different evaluation metrics while y-axis is for performance. Different color bars present different ratios of labeled data, where green bars are for 1%, blue bars are for 10%, and red bars are for 30%.

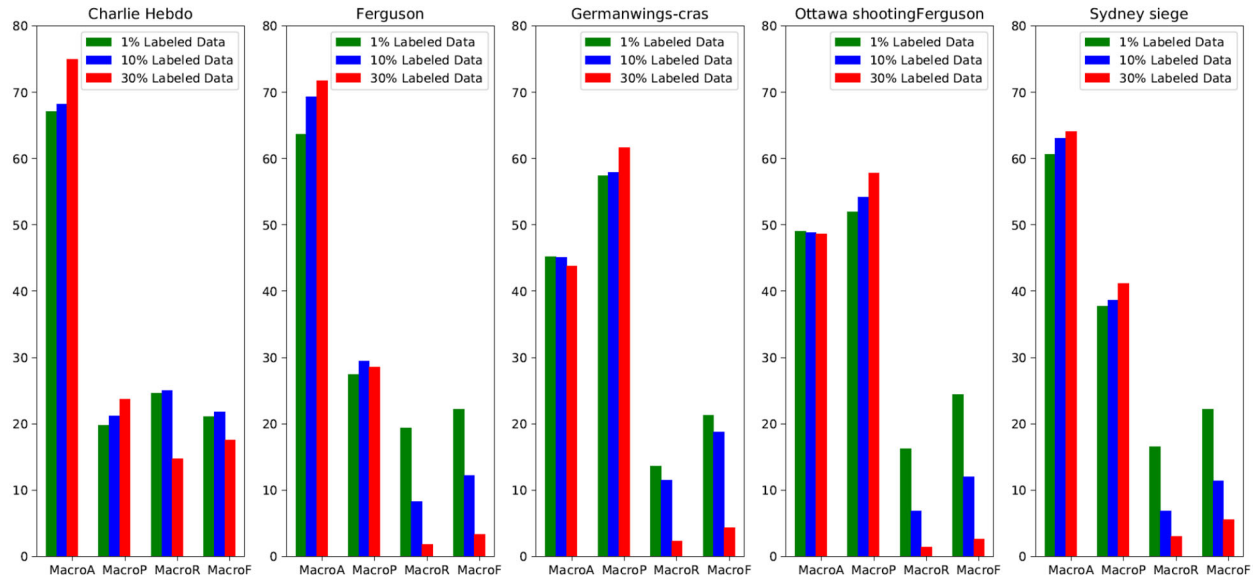


Figure 86: Comparing performance for the case of embedding size 256. x-axis is for different evaluation metrics while y-axis is for performance. Different color bars illustrate different ratios of labeled data, where green bars are for 1%, blue bars are for 10%, and red bars are for 30%.

4.4 Implementation, Visualization, and Validation

4.4.1 Implementation of real world applications in UAV tracking

4.4.1.1 Model Running on NVIDIA Jetson TX2

We build the ARM version of TensorFlow 1-4.1 and install it on NVIDIA Jetson TX2. The size of our model is about 1 GB and we test it with 10000 images (i.e. the validation dataset). Every inference, we apply 50 images and measure the spent time. The average fps is about 29. Since the memory in TX2 is 8 GB and shared by both CPUs and gpu, we have to specify how much memory the gpu uses. Here we set 0.3 (2.4 GB memory) to avoid the starvation of memory caused by the gpu. This is because running the gpu with too much memory would result in the performance degradation, even the system stuck. Figure 87 presents the results of our model with the DAC dataset. The green bounding box is the ground truth and the red one is the bounding box predicted by our model.

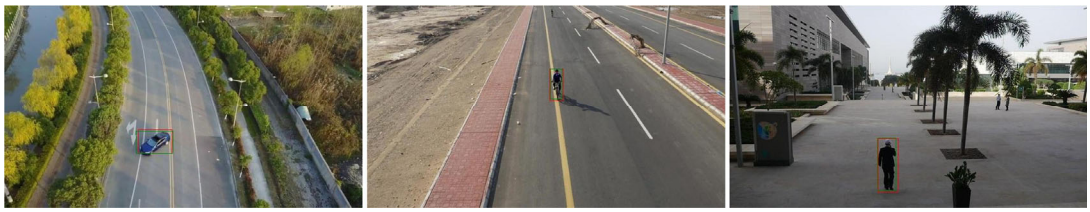


Figure 87: The results of running our model with the DAC dataset. The green bounding box is the ground truth and the red one is our prediction.

4.4.1.2 Experiments with Drone

We test the proposed algorithm on a DJI S1000 drone and tracked a moving white car successfully. The experiment is conducted in San Rafael park in Reno, Nevada, USA. The images are collected by a GoPro Hero 4 which is carried by a gimbal on the drone. We mount a NVIDIA Jetson TX2 on the drone to perform object detection and tracking using the proposed deep learning model. The video stream is transmitted from GoPro to TX2 via an external video card dongle. TX2 then perform inference based on received video stream and run the tracking algorithm. A control command is then generated and delivered from TX2 to the lower flight controller, which is a Pixhawk 4

drone controller. The drone controller will interpret the received control command and send control signals to the motors directly. Figure 88 shows the structure of the drone.

During the experiment, a white car is set as the target. The car is driven along a random path. The status of the drone is monitored by the ground control station. Figure 89 shows the experiment scene and the drone flies at 10 meter autonomously. We take four different videos to preserve the diversity, and use them as our dataset. All the images are annotated as either background or car with the corresponding bounding boxes. There are 21846 images in total and we use 2000 images from them as the validation dataset, where the image size is 720×1280. The coordinate of the bounding box for the background is (-1,-1,-1,-1) for simplicity. When calculating the iou of the background, we always regard it as 0. That is, we only consider the iou of the white car.

We do not change the architecture of our model in Figure 28 except the first *CONV*. Since the length and width of the images we used are double the length and width of the DAC dataset, the filter size and the stride would need to be changed to 12x20 and 4x6, respectively, to reduce the size of the image into the same dimension.

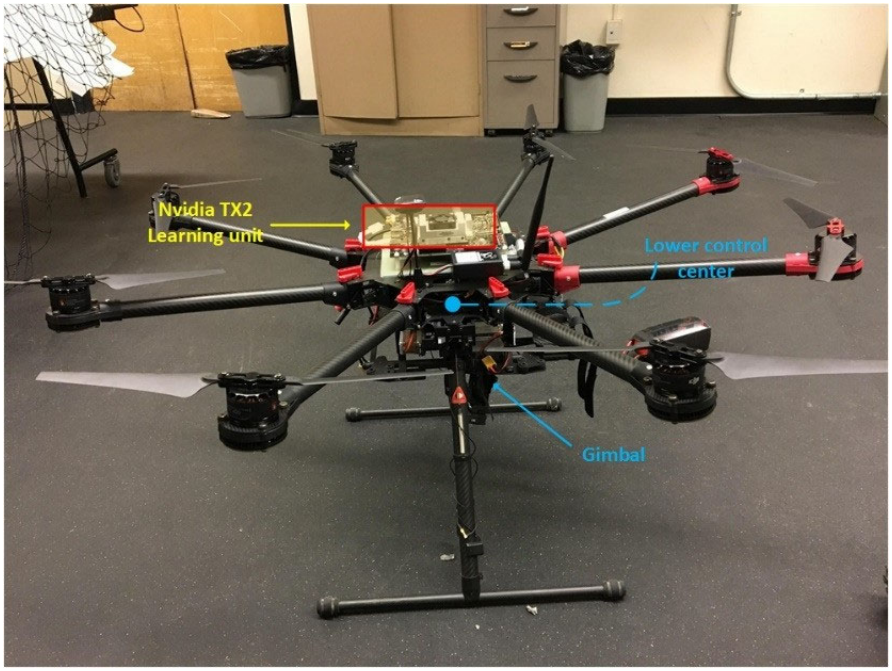


Figure 88: Illustration of our testing drone and its peripherals.

.

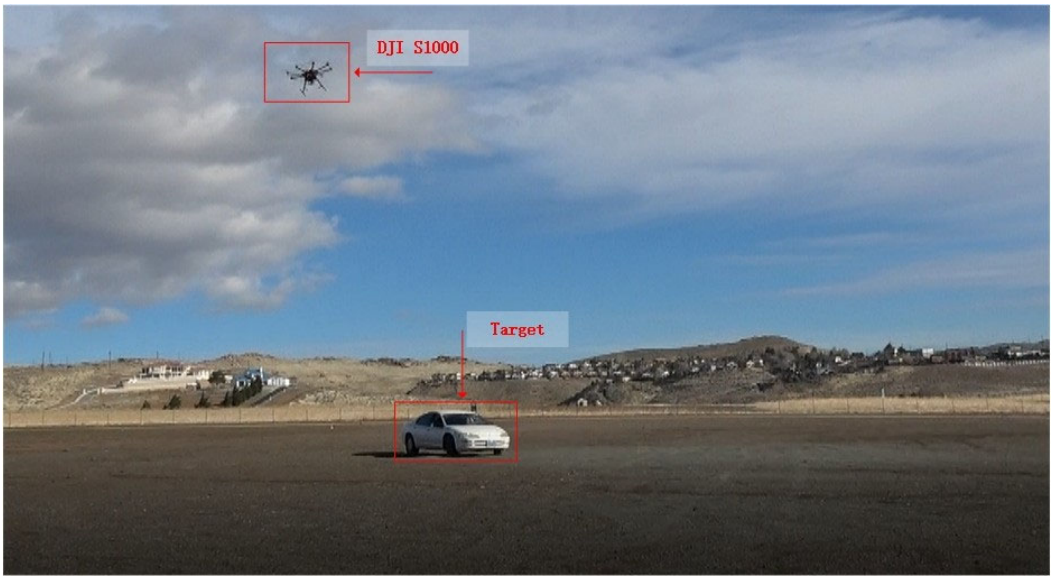


Figure 89: Scenario of the self-navigating drone using the single object detection.

To calculate the iou in a reasonable way, we don't take the iou of the background into the consideration, which makes no sense. Only the iou of the white car and the classification accuracy are important and considered. With the help of classification, we will be able to know when the object is not inside our view and have to take the other actions like spinning around or starting the GPS. Figure 90 shows two sample images.

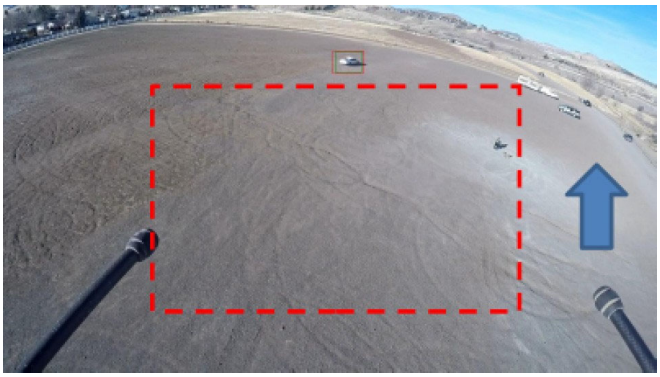
Figure 91 shows two examples of how the drone navigates itself with the help of our model. We set up a region that if the car appears inside it, the drone would stay around without taking any action. The size of the region is $\frac{2}{3}$ of the image size and centered in the image. If the size of the region is too large, the object would easily disappear from view. On the other hand, if the size is too small, the drone would keep controlling itself in a fine movement, which in turn causes the waste of the computing power and battery. Although Pixhawk is an independent open-hardware project that aims to provide the standard for readily-available, high-quality and low-cost autopilot hardware designs, Pixhawk might not be good enough to do the object tracking control. Since the tracking control is a type of adaptive control, we usually need other strong mini-PC to do it. Specifically, it is composed of two steps: path planning and tracking error reduction. The former gen-

erates a y trajectory that uav needs to follow using the detected object. With given the uav dynamics as a set of dynamic equations, the latter designs a tracking control to reduce the tracking error by making the difference between practical y path and desired y trajectory converge to zero. Both of these two steps require very high computing capability so it is hard for the embedded systems to meet the requirements.

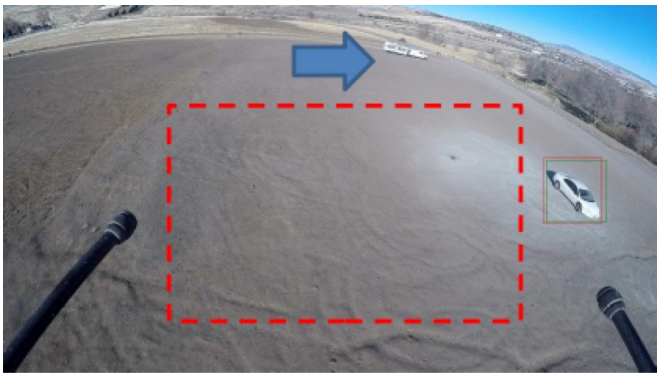
In Figure 92(a), the training curve of iou converges slower and less stable than the one of DAC dataset. We think it is because the scenarios of the car and background are very similar to each other that the model needs more time to learn how to differentiate either of them, and easily makes the mistake of the identification. That is, it increases the difficulty of the classification too. In addition, if the part of the car appears in the image, even occluded by the drone or the boundary of the image, we still assign the proper bounding box to it and label it as car (we don't see such case in the DAC dataset). As shown in Figure 92(b), we use exactly the same hyperparameters adopted in Section 3.4.3.1 but the training curves of the classification and iou push and pull each other severely. Even though the model with the higher iou can be trained, the classification accuracy is not quite satisfying. It is very critical that the classification accuracy plays the important role of identifying the background. As such, the weights of the weights of the softmax and mse we used in this experiment are 1.0 and $1e^{-3}$respectively to make the classification accuracy stable and high. Furthermore, the size of car varies a lot in our images so it increases the difficulty of detecting the car correctly. Due to these challenging issues, we use $1e^{-4}$ as our learning rate for the first 100 Epochs, and then 1e-5 to the end in our learning scheme. The classification accuracy is 99.7% and iou is 35.9%.

(a) The car before the drone flies. (b) The car when the drone is flying.

Figure 90: Two scenarios of the car used in the training.

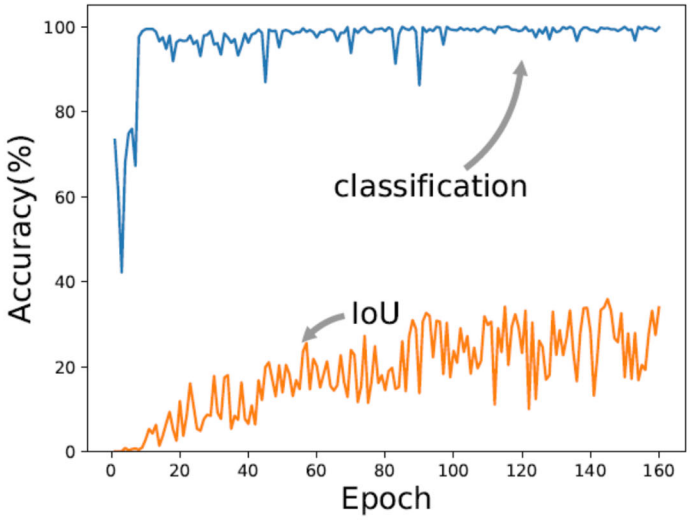


(a) The car is detected outside of the view so the drone would move forward and try to make the car detected inside the view.

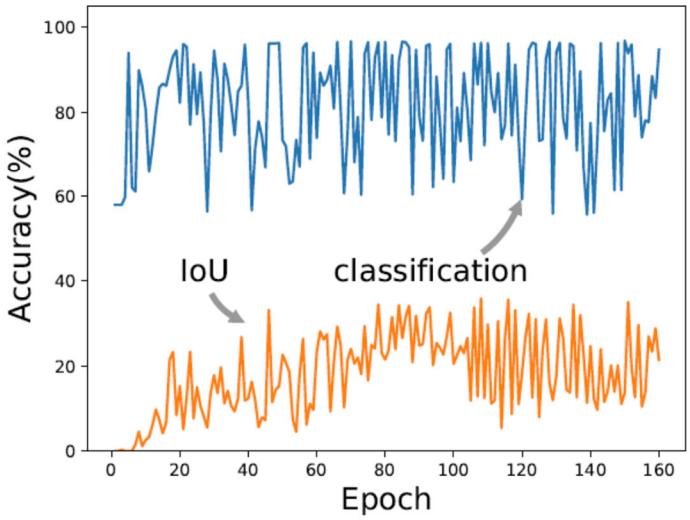


(b) The car is detected on the right hand side of the view so the drone would make left a little bit to follow the car.

Figure 91: Illustration of how the drone navigates itself using the information from object detection. If the car is detected inside the red rectangle, the drone just stays around. By contrast, if the car is outside of this view, the drone would navigate itself to make the car detected inside the view.



(a) The weights of the softmax and mse: 1.0 and 1e-3, respectively.



(b) The weights of the softmax and mse: 1e-2 and 1e-3, respectively.

Figure 92: The validation accuracy and IoU of our model trained with the dataset col-lected in Reno and the different weights.

4.4.2 Data Visualization on cloud

With the collaboration with Thermal Fisher Scientific, the student works in the project is able to understand the internal data structures and APIs of the Open Inventor toolkit. Data in Open Inventor is stored as an Octree structure in one file. In order to reduce the data loading time, we precomputes the LOD of any seismic files in SAC and convert the previous slice-based data storage into the brick format. We define the size of bricks and the index for each brick referring to the rules given by the Thermo Fisher Scientific.

In a standard seismic data file, the data is stored slice by slice. In order to transform these slices to bricks, we aggregate several slices at first according to the defined brick size. In this way, we get multiple large cuboids, which size in the inline direction is the same as the defined brick size. After that, we cut each cuboid along the other two directions to get the final bricks. At last, we define a 3D index as the start position for each brick. After creating the brick format, we create several levels of bricks in different resolutions. We subsample one level of bricks to generate a lower resolution brick. Figure 93 shows the Octree structure and level of details.

After we transform data stored in slices to bricks, we need to give a unique index to each brick following the rule given by the Thermo Fisher Scientific so that the Open Inventor can query each brick using its unique index. And all the bricks in each brick volume need to follow the order in Figure 94 to define their indices.

To further improve the visualization efficiency, we cache the LOD data in each file into memory at the beginning when a user accesses it. When the users try to view the seismic data next time, the system can directly query required bricks from the memory without invoking any calculation. Because the resource on each device is limited, we set up a limited number for cached data files. When there are more than five files cached in this array, the system is going to drop the first cached data and cache the new data instead.

After the brick format and LOD implementation, the data service and rendering service communication are improved. As we described before, SAC uses ZeroMQ and ProtoBuf to implement the data communication. When users interact with data view through the web portal, only partial bricks at a certain level of details are needed for the rendering service to complete its job. The rendered images are transferred to the users web browser to achieve real-time data visualization and interaction.

Table 25 shows the performance after we change the Open Inventor configuration from reading slices to reading bricks. In this test, we use a 198.7MB seismic volume to compare the performance. From this table, we can find using LOD and bricks helps a lot to display image faster. Figure 95 shows how images looked in different resolutions.

SAC was built mostly on top of open source software packages. For visualization, we have a choice of either selecting an open source visualization package or using the Open Inventor toolkit, which is proprietary software from Thermo Fisher Scientific. There are obvious pros and cons for open source and proprietary software. For our case, we collaborate with the Thermo Fisher Scientific since it is a good opportunity to our students to learn the industry standards. The Open Inventor toolkit has advantages of robustness and high-performance. By consulting with the experts in Thermo Fisher, we are able to understand Open Inventor toolkit and build the visualization module quickly. The academic team spent more time in the data analytics and scalability research, which has been the focus of the research team.

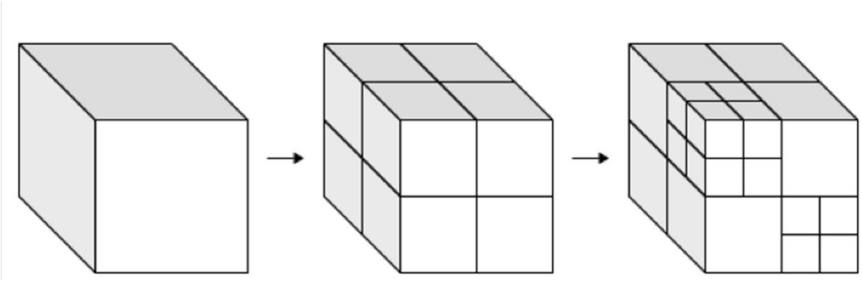


Figure 93: Octree structure used for the level of details.

Table 25: The image loading time for the same seismic data file.

	Full resolution	1/2 resolution	1/4 resolution	1/8 resolution
Read in Slices	19.11 second	12.48 second	6.52 second	5.03 second
Read in Brick	10.18 second	2.08 second	1.15 second	0.65 second

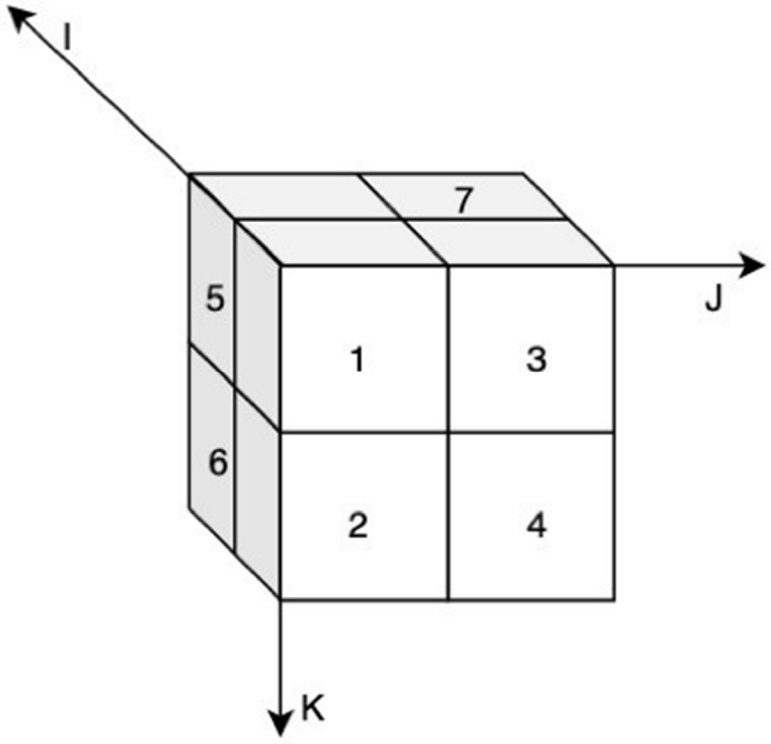


Figure 94: Index order in Octree structure.

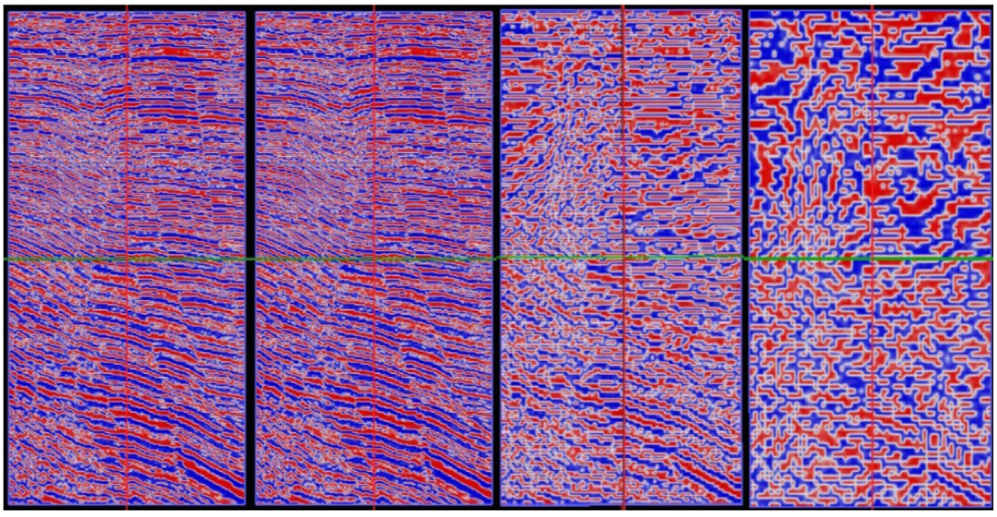


Figure 95: Image in different resolution (From left to right is full, 1/2, 1/4, 1/8 resolution).

5.0 CONCLUSIONS

This project has made significant technical contributions in all four research thrust areas. The details are given below.

5.1 Thrust 1: Big Data Cloud Computing System

In this research thrust, a customized domain specific big data analytics cloud for CREDIT research has been built. Cloud Computing as a disruptive technology, provides a dynamic, elastic, and promising computing climate to tackle the challenges of big data processing and analytics. Hadoop and MapReduce are the widely used open source frameworks in Cloud Computing for storing and processing big data in the scalable fashion. Spark is the latest parallel computing engine working together with Hadoop that exceeds MapReduce performance via its in-memory computing and high level programming features. In this project, we proposed a novel design and demonstrated implementation of a productive, domain-specific big data analytics cloud platform on top of Hadoop and Spark. To increase users' productivity, we created a variety of data processing templates to simplify the programming efforts. We have conducted experiments for its productivity and performance with a few basic but representative data processing algorithms in the petroleum industry. Geophysicists can use the platform to productively design and implement scalable seismic data processing algorithms without handling the details of data management and the complexity of parallelism. The Cloud platform generates a complete data processing application based on users' kernel program and simple configurations, allocates resources and executes it in parallel on top of Spark and Hadoop.

Furthermore, a concept of integrated High Performance Computing (HPC) state-of-the- art technology into big data analytics for performance and scale has been proposed. We have developed high-level APIs, compiler, and runtime solutions to enhance the efficiency of computing. The multidimensional array is a fundamental data structure that has been widely used in scientific computing, as well as in many big data analytics applications. Distributed multi-dimensional array has been well studied in the HPC platforms; however, little research has been done in the widely-used big data analytics platforms. In this project, we completed an implementation of Distributed Multidimensional Array Toolkit (DMAT) on top of the Apache Spark big data analytics platform. The

toolkit supports several fashions for multidimensional array distributions, repartition, transposition, access, and data parallelism with a variety of parallel execution templates. We also introduced the software architecture and implementations of DMAT, and studied the performance characteristics of some typical multi-dimensional array operations with different configurations.

5.2 Thrust 2: Reliable and Robust Data Collection and Aggregation

In this research thrust, edge computing has been studied extensively to support reliable and robust data collection and aggregation. Edge intelligent computing is an important emerging research topic with the deployment of 5G and IoT in recent years. At the same time, privacy preservation of users is indispensable. For image/video data, the proposed autoencoder based edge computing framework has better privacy preserving and security guarantee than federated learning, it is also less constrained by the limited computational resources of edge devices. The proposed framework encourages novel design and implementation of efficient privacy-preserving edge intelligent computing. It provides 1) flexibility of training autoencoder at each edge device individually, thus protect the data privacy of end-users because raw data is not transmitted at any time; 2) after the training of autoencoder is complete, raw data is \compressed and encrypted" by the encoder before transmitting to the edge server, and this will reduce the communications cost, and further protect the data privacy and security; 3) the autoencoder will provide features to the classifier at the server, thus allow the classifier to be trained on the features with less and controlled dimensions; 4) the decoupling of the training of the autoencoder at the edge devices and the training of the classifier at the edge server relaxes the frequent communications requirement between edge devices and edge server. Experiments have been carried out using CIFAR10 and ImageNet datasets. A detailed analysis of the tradeoff between classifier accuracy, the dimensionality of data, compression ratio, and various choices of classifiers has been given to provide benchmarks and insights on the proposed scheme. To the best of our knowledge, this is the first attempt to design a framework to address the image classification problem in an edge computing scenario, where an autoencoder is designed to compress the raw images and extract salient features at the same time. The proposed framework has been compared to the uncompressed approach (compression ratio = 1), which can be considered the baseline model. In addition, all the transfer learning models are indeed state-of-the-art. Combining them with the proposed framework will result in a highly efficient and privacy-preserving edge intelligent computing solution. In addition, computation offloading is an emerging

technology that has been investigated in this thrust. Specifically, multi-task learning has been applied to computation offloading optimization that reduce the inference time by 4-order of magnitude while achieving better accuracy. In this work, we propose a multi-task learning based feed-forward neural network (MTFNN) model to achieve an optimal computation offloading strategy for the mobile edge computing (MEC) system. We first formulate the joint optimization of binary offloading decision and computational resource allocation as a mixed integer nonlinear programming (MINLP) problem. Then, a MTFNN model is trained offline to solve the optimization problem with high accuracy. The pre-trained model can then directly infer the solution to the MINLP problem online with very low computational cost. The effects on the system performance from the inference error in the classification problem and the inference bias in the regression problem are analyzed and some implementation issues are discussed as well. Testing results show that the proposed MTFNN model outperforms the conventional optimization algorithms significantly in terms of computation time (four orders of magnitude) and inference accuracy (up to two times better).

5.3 Thrust 3: Knowledge Extraction using Machine Learning and Deep Learning

In this research thrust, the feasibility of using Dempster-Shafer Theory (DST) and Dezert-Smarandache Theory (DSmT) for big data processing has been explored and a detection framework to mitigate the e_ect of uncertainty using Evidence Theory (DST - DSmT) and Kullback - Leibler (KL) divergence for distance measures is proposed and studied. The combination rules in DSmT such as PCR5 have very high computational complexity when the number of hypotheses are large, thus they cannot be directly applied to multiple big data sources with high cardinality. We proposed models with reduced number of classes and thus smaller size of power set and hyper power set. It results in lower computational cost and we evaluate its performance through a case study. Specifically, the proposed methods are applied to analyze temperature and humidity data for smart building applications. To decrease the number of focal elements and improve the computation complexity, we considered the exclusivity between hypotheses to simplify the model. Furthermore, generating frame of discernment dynamically decreases the cardinality of singleton focal elements. Computational complexity of the proposed method is derived analytically. The results using both synthetic and real data sets demonstrate the potentials of the proposed method for big data processing when the data sets contain high level of uncertainty.

Furthermore, semi-supervised learning based methods have been studied for the case of limited labeled data. The proposed semi-supervised learning can obtain high inference accuracy using

even very limited labeled data, which is a promising solution for real-time ML applications. Specifically, we proposed a novel two-path deep semi-supervised learning (TDSL) framework containing three CNNs, where both labeled data and unlabeled data are used jointly to train the model and enhance the detection performance. We implemented a Word CNN based TDSL to detect fake news with limited labeled data and compare its performance with various deep learning based baselines. Moreover, we validated the implemented model by testing on the LIAR and PHEME datasets. It is observed that the proposed model detects fake news effectively even with extremely limited labeled data. The proposed framework could be applied to address other tasks. In addition, novel deep semi-supervised learning models can be implemented based on the proposed framework with various designs of CNNs, which will be determined by the intended applications and tasks.

5.4 Thrust 4: Implementation, Visualization, and Validation

In this research thrust, a novel multi-task learning based deep learning model is designed and tested for object identification and target tracking on UAVs. It achieved real-time processing (> 20 fps) and high IoU (> 60%). Specifically, we proposed a novel modeling method of CNN suitable for the missions that require real-time processing of images/video on the UAV. We point out that the characteristics of the data and the requirements of the mission might vary a lot, so a general purpose model is hard to satisfy the requirements. Instead we integrate the essential concepts of RCNN and YOLO into the construction of our proposed model. To increase the efficiency of our model, we proposed the inducing layer to optimize the model and speed up the convergence in a stable way. With the dataset of the UAVs provided by DAC 2018, we demonstrated how to develop and train the model. The performance analysis shows our modeling method works well on the dataset: about 90% accuracy of the classification (98 classes) and 60% accuracy of IoU. We also performed field experiments with our dataset collected from the drone at 10-meter height. Running on the NVIDIA TX2 mounted on the UAV, the model has about 100% accuracy of the classification (car and background) and about 36% accuracy of IoU, which is good enough for object detection and tracking. In addition, a cloud-based big data visualization system has been built and achieved real-time data visualization on cloud. The collaboration between Prairie View A&M University and Thermo Fisher Scientific delivered a successful research platform that combines the power of scalable big data analytics and a close-to real-time big 3D data visualization capability. All of these

functionalities are delivered to end users via a cloud platform and accessible via a web-based application.

In sum, the research carried out at the CREDIT center results in an effective and efficient solution to the big data analysis needs of the DOD. Leveraging many promising technologies in artificial intelligence (AI) and machine learning (ML) especially deep learning (DL), the proposed solution will revolutionize the big data processing field of study. With the strong support of the government agencies especially DOD and our academic and industrial partners, the team is confident that the CREDIT center will further improve its research and education capacity and continue to train students especially underrepresented minorities to be highly qualified workforce and contribute to DOD missions and the nation for years to come.

6.0 REFERENCES

[1] DARPA. *DARPA-BAA-10-94: Insight*. 2010.

[2] National Research Council. *Frontiers in Massive Data Analysis* National Academies Press, Washington, DC, USA, 2013.

[3] Judith Bayard Cushing. Beyond big data? *Computing in Science and Engineering*, 15(5):4–5, 2013.

[4] H. Adolfy F.J. Alexander and A. Szalay. Special issue on big data. *Computing in Science Engineering*, 13(6), 2011.

[5] J.B. Cushing and J. French. Special issue on science data management. *Computing in Science Engineering*, 15(3), 2013.

[6] Message Passing Interface Forum. http://www.mpi-forum.org.

[7] {O}pen{MP}: Simple, Portable, Scalable {SMP}Programming. \url{http://www.openmp.org}, 2006.

[8] Tarek El-Ghazawi, William Carlson, Thomas Sterling, and Katherine Yelick. *UPC: Distributed Shared Memory Programming*. John Wiley and Sons, May 2005.

[9] Robert W Numrich and John Reid. Co-array {Fortran} for parallel programming. *SIGPLAN Fortran Forum*, 17(2):1–31, 1998.

[10] Philippe Charles, Christopher Donawa, Kemal Ebcioglu, Christian Grothoff, Allan Kielstra, Vijay Saraswat, Vivek Sarkar, and Christoph Von Praun. *{X10}*: An Object-Oriented Approach to Non-Uniform Cluster Computing. In *Proceedings of the 20th {ACM SIGPLAN} conference on Object-oriented programing, systems, languages, and applications*, pages 519–538. ACM SIGPLAN, 2005.

[11] J. Dean S. Ghemawat. MapReduce: simplified data processing on large clusters. In *Communications of the ACM - 50th anniversary issue: 1958 - 2008*, volume 51, pages 107–113. ACM New York, January 2008.

[12] Hadoop Introduction. http://hadoop.apache.org/. [Retrieved: January, 2014].

[13] Storm: Distributed and fault-tolerant realtime computation. https://storm.incubator.apache.org//. [Retrieved: January, 2014].

[14] Apache Hadoop database, a distributed, scalable, big data store. http://hbase.apache.org/. [Retrieved: January, 2014].

[15] S4: distributed stream computing platform. http://incubator.apache.org/s4/. [Retrieved: May, 2014].

[16] Esper: Complex Event Processing. http://esper.codehaus.org/. [Retrieved: May, 2014].

[17] StreamBase: Complex Event Processing System. http://www.streambase.com/. [Retrieved: May, 2014].

[18] Avinash Lakshman and Prashant Malik. Cassandra: A decentralized structured storage system. *SIGOPS Oper. Syst. Rev.*, 44(2):35–40, April 2010.

[19] Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, and Ion Stoica. Spark: Cluster computing with working sets. In *Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing*, HotCloud'10, pages 10–10, Berkeley, CA, USA, 2010. USENIX Association.

[20] Matei Zaharia, Tathagata Das, Haoyuan Li, Scott Shenker, and Ion Stoica. Dis- cretized streams: An efficient and fault-tolerant model for stream processing on large clusters. In *Proceedings of the 4th USENIX Conference on Hot Topics in Cloud Ccomputing*, HotCloud'12, pages 10–10, Berkeley, CA, USA, 2012. USENIX Association.

[21] Matei Zaharia Mosharaf Chowdhury and Tathagata Das. Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing. In *NSDI'12 Proceedings of the 9th USENIX Conference on Networked Systems De- sign and Implementation*, San Jose, CA, April 2012. USENIX Association Berke- ley.

[22] Gunho Lee, Byung-Gon Chun, and Randy H Katz. Heterogeneity-aware resource allocation and scheduling in the cloud. *Proceedings of HotCloud*, pages 1–5, 2011.

[23] Gunho Lee. *Resource allocation and scheduling in heterogeneous cloud environments*. University of California, Berkeley, 2012.

[24] Benjamin Hindman, Andy Konwinski, Matei Zaharia, Ali Ghodsi, Anthony D Joseph, Randy H Katz, Scott Shenker, and Ion Stoica. Mesos: A platform for fine-grained resource sharing in the data center. In *NSDI*, volume 11, pages 22–22, 2011.

[25] Jorda Polo, David Carrera, Yolanda Becerra, Vicenç Beltran, Jordi Torres, and Eduard Ayguadé. Performance management of accelerated mapreduce workloads in heterogeneous clusters. In *Parallel Processing (ICPP), 2010 39th International Conference on*, pages 653–662. IEEE, 2010.

[26] Matei Zaharia, Andy Konwinski, Anthony D Joseph, Randy H Katz, and Ion Stoica. Improving mapreduce performance in heterogeneous environments. In *OSDI*, volume 8, page 7, 2008.

[27] Ganesh Ananthanarayanan, Srikanth Kandula, Albert G Greenberg, Ion Stoica, Yi Lu, Bikas Saha, and Edward Harris. Reining in the outliers in map-reduce clusters using mantri. In *OSDI*, volume 10, page 24, 2010.

[28] YongChul Kwon, Magdalena Balazinska, Bill Howe, and Jerome Rolia. Skew- resistant parallel processing of feature-extracting scientific user-defined functions. In *Proceedings of the 1st ACM symposium on Cloud computing*, pages 75–86. ACM, 2010.

[29] Chao Tian, Haojie Zhou, Yongqiang He, and Li Zha. A dynamic mapreduce scheduler for heterogeneous workloads. In *Grid and Cooperative Computing, 2009. GCC'09. Eighth International Conference on*, pages 218–224. IEEE, 2009.

[30] Georg Dotzler, Ronald Veldema, and Michael Klemm. Jcudamp: Openmp/java on cuda. In *Proceedings of the 3rd International Workshop on Multicore Software Engineering*,

pages 10–17. ACM, 2010.

[31] Michael Klemm, Matthias Bezold, Ronald Veldema, and Michael Philippsen. Jamp: an implementation of openmp for a java dsm. *Concurrency and Com- putation: Practice and Experience*, 19(18):2333–2352, 2007.

[32] Yonghong Yan, Max Grossman, and Vivek Sarkar. Jcuda: A programmer-friendly interface for accelerating java programs with cuda. In *Euro-Par 2009 Parallel Processing*, pages 887–899. Springer, 2009.

[33] Sergio Barrachina, Maribel Castillo, Francisco D Igual, Rafael Mayo, and En- rique S Quintana-Orti. Evaluation and tuning of the level 3 cublas for graphics processors. In *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, pages 1–8. IEEE, 2008.

[34] Michael D McCool and Stefanus Du Toit. *Metaprogramming GPUs with Sh*. AK Peters Wellesley, 2004.

[35] Jens Breitbart. Cupp-a framework for easy cuda integration. In *Parallel & Dis- tributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*, pages 1–8. IEEE, 2009.

[36] Guodong Han, Chenggang Zhang, King Tin Lam, and Cho-Li Wang. Java with auto-parallelization on graphics coprocessing architecture. In *Parallel Processing (ICPP), 2013 42nd International Conference on*, pages 504–509. IEEE, 2013.

[37] Saman Amarasinghe, Michael I Gordon, Michal Karczmarek, Jasper Lin, David Maze, Rodric M Rabbah, and William Thies. Language and compiler design for streaming applications. *Int. J. Parallel Program,* 33(2):261–278, 2005.

[38] Ian Buck, Tim Foley, Daniel Horn, Jeremy Sugerman, Kayvon Fatahalian, Mike Houston, and Pat Hanrahan. Brook for {GPU}s: stream computing on graphics hardware. In SIGGRAPH *'04: ACM SIGGRAPH 2004 Papers*, pages 777–786, New York, NY, USA,

2004. ACM.

[39] Dan Zhang, Zeng Z Li, Hong Song, and Long Liu. A Programming Model for an Embedded Media Processing Architecture. In *SAMOS: Embedded Computer Systems: Architectures, Modeling, and Simulation*, volume 3553 of *Lecture Notes in Computer Science*, pages 251–261, 2005.

[40] William R Mark, R Steven Glanville, Kurt Akeley, and Mark J Kilgard. Cg: a system for programming graphics hardware in a C-like language. In *SIGGRAPH '03: ACM SIGGRAPH 2003 Papers*, pages 896–907, New York, NY, USA, 2003. ACM.

[41] Michael K Chen, Xiao Feng Li, Ruiqi Lian, Jason H Lin, Lixia Liu, Tao Liu, and Roy Ju. Shangri-La: achieving high performance from compiled network applica- tions while enabling ease of programming. In *PLDI '05: Proceedings of the 2005 ACM SIGPLAN conference on Programming language design and implementa- tion*, pages 224–236, New York, NY, USA, 2005. ACM.

[42] Charles Consel, Hedi Hamdi, Laurent Réveillère, Lenin Singaravelu, Haiyan Yu, and Calton Pu. Spidle: a DSL approach to specifying streaming applications. In *GPCE '03: Proceedings of the 2nd international conference on Generative programming and component engineering*, pages 1–17, New York, NY, USA, 2003. Springer-Verlag New York, Inc.

[43] M Gonzalez, E Ayguade, X Martorell, and J Labarta. Exploiting pipelined ex- ecutions in OpenMP. In *International Conference on Parallel Processing*, pages 153–160, 2003.

[44] Jesper H Spring, Jean Privat, Rachid Guerraoui, and Jan Vitek. Streamflex: high-throughput stream programming in java. *SIGPLAN Not.,* 42(10):211–228, 2007.

[45] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. *Commun. ACM*, 51(1):107113, Jan 2 0 0 8 .

[46] Sameer Agarwal, Barzan Mozafari, Aurojit Panda, Henry Milner, Samuel Mad- den, and

Ion Stoica. Blinkdb: Queries with bounded errors and bounded response times on very large data. In *Proceedings of the 8th ACM European Conference on Computer Systems*, EuroSys '13, pages 29–42, New York, NY, USA, 2013. ACM.

[47] Amir Erfan Eshratifar, Mohammad Saeed Abrishami, and Massoud Pedram. Jointdnn: An efficient training and inference engine for intelligent mobile cloud computing services. *IEEE Transactions on Mobile Computing*, 20:565–576, 2021.

[48] Wenqi Shi, Yunzhong Hou, Sheng Zhou, Z. Niu, Y. Zhang, and L. Geng. Improv- ing device-edge cooperative inference of deep learning via 2-step pruning. *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 1–6, 2019.

[49] Yiping Kang, Johann Hauswald, Cao Gao, Austin Rovinski, Trevor Mudge, Jason Mars, and Lingjia Tang. Neurosurgeon: Collaborative intelligence between the cloud and mobile edge. *SIGARCH Comput. Archit. News*, 45(1):615629, April 2017.

[50] J. Shao and J. Zhang. Bottlenet++: An end-to-end approach for feature compres- sion in device-edge co-inference systems. *2020 IEEE International Conference on Communications Workshops (ICC Workshops)*, pages 1–6, 2020.

[51] Amir Erfan Eshratifar, A. Esmaili, and Massoud Pedram. Bottlenet: A deep learning architecture for intelligent mobile cloud computing services. *2019 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, pages 1–6, 2019.

[52] Hyomin Choi and I. Bajic. Near-lossless deep feature compression for collabora- tive intelligence. *2018 IEEE 20th International Workshop on Multimedia Signal Processing (MMSP)*, pages 1–6, 2018.

[53] Pengtao Xie, Mikhail Bilenko, Tom Finley, Ran Gilad-Bachrach, Kristin E. Lauter, and Michael Naehrig. Crypto-nets: Neural networks over encrypted data. *ArXiv*, abs/1412.6181, 2014.

[54] Cynthia Dwork. Differential privacy: A survey of results. In Manindra Agrawal, Dingzhu Du, Zhenhua Duan, and Angsheng Li, editors, *Theory and Applications of Models of Computation*, pages 1–19, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[55] Martin Abadi, Andy Chu, Ian J. Goodfellow, H. Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep learning with differential privacy. *ArXiv*, abs/1607.00133, 2016.

[56] Jian Liu, Mika Juuti, Yao Lu, and N. Asokan. Oblivious neural network predic- tions via minionn transformations. *IACR Cryptology ePrint Archive*, 2017:452, 2017.

[57] Yifan Tian, Jiawei Yuan, and Houbing Song. Efficient privacy-preserving authen- tication framework for edge-assisted internet of drones. *Journal of Information Security and Applications*, 48:102354, 2019.

[58] Jie Li, Huang Lu, and Mohsen Guizani. Acpn: A novel authentication frame- work with conditional privacy-preservation and non-repudiation for vanets. *IEEE Transactions on Parallel and Distributed Systems*, 26(4):938–948, 2015.

[59] Chien-Ming Chen, Lili Chen, Yanyu Huang, Sachin Kumar, and Jimmy Ming-Tai Wu. Lightweight authentication protocol in edge-based smart grid environment. *EURASIP Journal on Wireless Communications and Networking*, 68, 2021.

[60] Prosanta Gope and Biplab Sikdar. An efficient privacy-preserving authenticated key agreement scheme for edge-assisted internet of drones. *IEEE Transactions on Vehicular Technology*, 69(11):13621–13630, 2020.

[61] Yunru Zhang, Debiao He, Li Li, and Biwen Chen. A lightweight authentication and key agreement scheme for internet of drones. *Computer Communications*, 154:455–464, 2020.

[62] Prosanta Gope. Pmake: Privacy-aware multi-factor authenticated key establishment scheme for advance metering infrastructure in smart grid. *Computer Communications*,

152:338–344, 2020.

[63] Jimmy Ming-Tai Wu, Justin Zhan, and Jerry Chun-Wei Lin. Ant colony system sanitization approach to hiding sensitive itemsets. *IEEE Access*, 5:10024–10039, 2017.

[64] Jerry Chun-Wei Lin, Gautam Srivastava, Yuyu Zhang, Youcef Djenouri, and Moayad Aloqaily. Privacy-preserving multiobjective sanitization model in 6g iot environments. *IEEE Internet of Things Journal*, 8(7):5340–5349, 2021.

[65] N-sanitization: A semantic privacy-preserving framework for unstructured medi- cal datasets. *Computer Communications*, 161:160–171, 2020.

[66] Jerry Chun-Wei Lin, Jimmy Ming-Tai Wu, Philippe Fournier-Viger, Youcef Dje- nouri, Chun-Hao Chen, and Yuyu Zhang. A sanitization approach to secure shared data in an iot environment. *IEEE Access*, 7:25359–25368, 2019.

[67] Jerry Chun-Wei Lin, Qiankun Liu, Philippe Fournier-Viger, Tzung-Pei Hong, Miroslav Voznak, and Justin Zhan. A sanitization approach for hiding sensitive itemsets based on particle swarm optimization. *Engineering Applications of Artificial Intelligence*, 53:1–18, 2016.

[68] Jimmy Ming-Tai Wu, Gautam Srivastava, Unil Yun, Shahab Tayeb, and Jerry Chun-Wei Lin. An evolutionary computation-based privacy-preserving data mining model under a multithreshold constraint. *Transactions on Emerging Telecom- munications Technologies*, 32(3):e4209, 2021.

[69] Ho Bae, Jaehee Jang, Dahuin Jung, Hyemi Jang, Heonseok Ha, and Sungroh Yoon. Security and privacy issues in deep learning. *ArXiv*, abs/1807.11655, 2018.

[70] J. Zhao, Y. Chen, and W. Zhang. Differential privacy preservation in deep learn- ing: Challenges, opportunities and solutions. *IEEE Access*, 7:48901–48911, 2019.

[71] Reza Shokri and Vitaly Shmatikov. Privacy-preserving deep learning. In *Proceed- ings of*

the 22nd ACM SIGSAC Conference on Computer and Communications Security, CCS 15, page 13101321, New York, NY, USA, 2015. Association for Computing Machinery.

[72] H. Brendan McMahan, Eider Moore, Daniel Ramage, and Blaise Agüera y Ar- cas. Federated learning of deep networks using model averaging. *CoRR*, abs/1602.05629, 2016.

[73] Timothy Yang, Galen Andrew, Hubert Eichner, Haicheng Sun, Wei Li, Nicholas Kong, Daniel Ramage, and Françoise Beaufays. Applied federated learning: Im- proving google keyboard query suggestions. *CoRR*, abs/1812.02903, 2018.

[74] Andrew Hard, Kanishka Rao, Rajiv Mathews, Françoise Beaufays, Sean Augen- stein, Hubert Eichner, Chloé Kiddon, and Daniel Ramage. Federated learning for mobile keyboard prediction. *CoRR*, abs/1811.03604, 2018.

[75] Mingqing Chen, Rajiv Mathews, Tom Ouyang, and Françoise Beaufays. Federated learning of out-of-vocabulary words. *CoRR*, abs/1903.10635, 2019.

[76] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Marcaurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, Quoc V. Le, and Andrew Y. Ng. Large scale distributed deep networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, Advances *in Neural Informa- tion Processing Systems 25*, pages 1223–1231. Curran Associates, Inc., 2012.

[77] Jianmin Chen, Rajat Monga, Samy Bengio, and Rafal Józefowicz. Revisiting distributed synchronous SGD. *CoRR*, abs/1604.00981, 2016.

[78] Hangyu Zhu and Yaochu Jin. Multi-objective evolutionary federated learning. *CoRR*, abs/1812.07478, 2018.

[79] Jakub Konecny, H. Brendan McMahan, Felix X. Yu, Peter Richtarik, Ananda Theertha Suresh, and Dave Bacon. Federated learning: Strategies for improving communication efficiency. In *NIPS Workshop on Private Multi-Party Machine Learning*, 2016.

[80] Ligeng Zhu, Zhijian Liu, and Song Han. Deep leakage from gradients. *CoRR*,

abs/1906.08935, 2019.

[81] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H. Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for privacy-preserving machine learning. In *CCS '17*, 2017.

[82] Yue Zhao, Meng Li, Liangzhen Lai, Naveen Suda, Damon Civin, and Vikas Chan- dra. Federated learning with non-iid data. *CoRR*, abs/1806.00582, 2018.

[83] Virginia Smith, Chao-Kai Chiang, Maziar Sanjabi, and Ameet Talwalkar. Federated multi-task learning. *CoRR*, abs/1705.10467, 2017.

[84] Praneeth Vepakomma, Tristan Swedish, R. Raskar, Otkrist Gupta, and Abhi- manyu Dubey. No peek: A survey of private distributed deep learning. *ArXiv*, abs/1812.03288, 2018.

[85] Otkrist Gupta and R. Raskar. Distributed learning of deep neural network over multiple agents. *J. Netw. Comput. Appl.*, 116:1–8, 2018.

[86] C. Thapa, M.A.P. Chamikara, and S. Camtepe. Splitfed: When federated learning meets split learning. *ArXiv*, abs/2004.12088, 2020.

[87] Iker Ceballos, Vivek Sharma, E. Múgica, Abhishek Singh, A. Román, Praneeth Vepakomma, and Ramesh Raskar. Splitnn-driven vertical partitioning. *ArXiv*, abs/2008.04137, 2020.

[88] Praneeth Vepakomma, Otkrist Gupta, Tristan Swedish, and Ramesh Raskar. Split learning for health: Distributed deep learning without sharing raw patient data. *ArXiv*, abs/1812.00564, 2018.

[89] Maarten G. Poirot, Praneeth Vepakomma, K. Chang, J. Kalpathy Cramer, R. Gupta, and R. Raskar. Split learning for collaborative deep learning in health- care. *ArXiv*, abs/1912.12115, 2019.

[90] Praneeth Vepakomma, Otkrist Gupta, Abhimanyu Dubey, and R. Raskar. Re- ducing

leakage in distributed deep learning for sensitive health data. 2019.

[91] Yansong Gao, M. Kim, Sharif Abuadbba, Yeonjae Kim, C. Thapa, Kyu yeon Kim, R. amtepe, Hyoungshick Kim, and S. Nepal. End-to-end evaluation of federated learning and split learning for internet of things. *2020 International Symposium on Reliable Distributed Systems (SRDS)*, pages 91–100, 2020.

[92] Vahid Mirjalili, Sebastian Raschka, Anoop Namboodiri, and Arun Ross. Semi- adversarial networks: Convolutional autoencoders for imparting privacy to face images. *2018 International Conference on Biometrics (ICB)*, Feb 2018.

[93] Rasim M. Alguliyev, Ramiz M. Aliguliyev, and Fargana J. Abdullayeva. Privacy- preserving deep learning algorithm for big personal data analysis. *Journal of Industrial Information Integration*, 15:1 – 14, 2019.

[94] Marcus D'Souza, Matthew Johnson, Jonas Dorn, Caspar Van Munster, Manuela Diederich, Christian Kamm, Saskia Steinheimer, Kristina Kravalis, Jacques Boisvert, Ian Ormesher, Lorcan Walsh, Abigail Sellen, Frank Dahlke, Bernard Uitdehaag, and Ludwig Kappos. Autoencoder - a new method for keeping data privacy when analyzing videos of patients with motor dysfunction (p4.001). *Neu- rology*, 90(15 Supplement), 2018.

[95] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief. A survey on mobile edge computing: The communication perspective. *IEEE Commun. Surveys & Tutorials*, 19(4):2322–2358, August 2017.

[96] Y. Li, J. Liu, B. Cao, and C. Wang. Joint optimization of radio and virtual machine resources with uncertain user demands in mobile cloud computing. *IEEE Transactions on Multimedia*, 20(9):2427–2438, Sep 2018.

[97] T. Taleb, K. Samdanis, B. Mada, H. Flinck, S. Dutta, and D. Sabella. On multi- access edge computing: A survey of the emerging 5g network edge cloud architec- ture and orchestration. *IEEE Communications Surveys & Tutorials*, 19(3):1657– 1681, May 2017.

[98] B. Yang, X. Cao, X. Li, Q. Zhang, and L. Qian. *Mobile Edge Computing based Hierarchical Machine Learning Tasks Distribution for IIoT*. IEEE Internet of Things Journal, Early Access, 2020.

[99] S. Barbarossa, S. Sardellitti, and P. D. Lorenzo. Communicating while computing: Distributed mobile cloud computing over 5g heterogeneous networks. *IEEE Signal Processing Magazine*, 31(6):45–55, November 2014.

[100] H. Guo, J. Liu, and J. Zhang. Efficient computation offloading for multi-access edge computing in 5g hetnets. In *IEEE International Conference on Communi- cations (ICC)*, Kansas City, MO, May 2018. Proc.

[101] Y. Zhang, D. Niyato, and P. Wang. Offloading in mobile cloudlet systems with intermittent connectivity. *IEEE Transactions on Mobile Computing*, 14(12):2516– 2529, February 2015.

[102] H. Eom, P. S. Juste, R. Figueiredo, O. Tickoo, R. Illikkal, and R. Iyer. Ma- chine learning-based runtime scheduler for mobile offloading framework. In *Proc. IEEE/ACM International Conference on Utility and Cloud Computing (UCC), , DC*, December 2013.

[103] S. Yu, X. Wang, and R. Langar. Computation offloading for mobile edge com- puting: a deep learning approach. In *IEEE Personal, Indoor, and Mobile Radio Communications (PIMRC)*, Montreal, QC, October 2017. Proc.

[104] T. X. Tran and D. Pompili. Joint task offloading and resource allocation for multi-server mobile-edge computing networks. *IEEE Transactions on Vehicular Technology*, 68(1):856–868, 2019.

[105] J. Zhang, X. Hu, and Z. Ning. *et al*. "Joint Resource Allocation for Latency- Sensitive Services over Mobile Edge Computing Networks with Caching, " *IEEE Internet of Things Journal*, Early Access, 2018.

[106] A. Khalili, S. Zarandi, and M. Rasti. *Joint Resource Allocation and Offload- ing Decision*

in Mobile Edge Computing. IEEE Communications Letters, Early Access, 2019.

[107] P. Zhao, H. Tian, C. Qin, and G. Nie. Energy-saving offloading by jointly allocating radio and computational resources for mobile edge computing. *IEEE Access*, 5:11255–11268, June 2017.

[108] H. Xing, L. Liu, J. Xu, and A. Nallanathan. Joint task assignment and wireless resource allocation for cooperative mobile-edge computing. In *IEEE International Conference on Communications (ICC)*, Kansas City, MO, May, 2018. Proc.

[109] F. Guo, H. Zhang, H. Ji, X. Li, and V. C. Leung. Energy efficient computation offloading for multi-access mec enabled small cell networks. In *IEEE International Conference on Communications (ICC) Workshops*, MO, May, 2018. Kansas City.

[110] Etsi Group Specification. Mobile edge computing (mec); technical requirements. *ETSI GS MEC 002 V1.1.1 (2016-03)*, March 2016.

[111] Etsi Group Specification. Mobile edge computing (mec); framework and reference architecture. *ETSI GS MEC 003 V1.1.1 (2016-03)*, Mar 2016.

[112] R. Ramesh. Predictive analytics for banking user data using aws machine learning cloud service. In *2017 2nd International Conference on Computing and Commu- nications Technologies (ICCCT)*, pages 210–215, Feb 2017.

[113] Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. Federated machine learning: Concept and applications. *ACM Trans. Intell. Syst. Technol.*, 10(2), January 2019.

[114] Robin C. Geyer, Tassilo Klein, and Moin Nabi. Differentially private federated learning: A client level perspective. *CoRR*, abs/1712.07557, 2017.

[115] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H. Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for federated learning on user-held data. In *NIPS Workshop on Private Multi-Party Machine Learning*, 2016.

[116] S. Burer and A. N. Letchford. Non-convex mixed-integer nonlinear programming: A survey. *Surveys in Operations Research and Management Science*, 17(2):97– 106, July 2012.

[117] J. Zhang and X. Hu. et al., "joint resource allocation for latency-sensitive services over mobile edge computing networks with caching,". *IEEE Internet of Things Journal*, 6(3):4283–4294, June 2019.

[118] T. X. Tran and D. Pompili. Joint task offloading and resource allocation for multi-server mobile-edge computing networks. *IEEE Transactions on Vehicular Technology*, 68(1):856–868, January 2019.

[119] S. Guo, J. Liu, Y. Yang, B. Xiao, and Z. Li. Energy-efficient dynamic computation offloading and cooperative task scheduling in mobile cloud computing. *IEEE Transactions on Mobile Computing*, 18(2):319–333, February 2019.

[120] H. Guo and J. Liu. Collaborative computation offloading for multiaccess edge computing over fiber-wireless networks. *IEEE Transactions on Vehicular Technology*, 67(5):4514–4526, May 2018.

[121] S. Joilo and G. Dán. Selfish decentralized computation offloading for mobile cloud computing in dense wireless networks. *IEEE Transactions on Mobile Computing*, 18(1):207–220, January 2019.

[122] T. K. Rodrigues, K. Suto, H. Nishiyama, J. Liu, and N. Kato. *Machine learning meets computation and communication control in evolving edge and cloud: Chal- lenges and future perspective*. IEEE Communications Surveys & Tutorials, Early Access, 2019.

[123] Y. S. Abu-Mostafa, M. Magdon-Ismail, and H. T. Lin. Learning from data. *New York, NY, USA:: AMLBook*, 4, March 2012.

[124] Google Edge Tpu. https://coral.ai/docs/edgetpu/faq/.

[125] Jetson Nano Developer Kit. https://developer.nvidia.com/embedded/jetson- nano-

developer-kit.

[126] Marco Maggipinto, Chiara Masiero, Alessandro Beghi, and Gian Antonio Susto. A convolutional autoencoder approach for feature extraction in virtual metrol- ogy. *Procedia Manufacturing*, 17:126 – 133, 2018. 28th International Conference on Flexible Automation and Intelligent Manufacturing (FAIM2018), June 11-14, 2018, Columbus, OH, USAGlobal Integration of Intelligent Manufacturing and Smart Industry for Good of Humanity.

[127] Ian J. Goodfellow, Yoshua Bengio, and Aaron C. Courville. Deep learning. *Nature*, 521:436–444, 2015.

[128] D. Adesina, J. Bassey, and L. Qian. Robust deep radio frequency spectrum learning for future wireless communications systems. *IEEE Access*, 8:148528– 148540, 2020.

[129] Z. Yu, E. Tan, D. Ni, J. Qin, S. Chen, S. Li, B. Lei, and T. Wang. A deep convolutional neural network-based framework for automatic fetal facial standard plane recognition. *IEEE Journal of Biomedical and Health Informatics*, 22(3):874–885, 2018.

[130] Doyen Sahoo, Quang Pham, Jing Lu, and Steven C. H. Hoi. Online deep learning: Learning deep neural networks on the fly. *CoRR*, abs/1711.03705, 2017.

[131] Heitor Murilo Gomes, Jesse Read, Albert Bifet, Jean Paul Barddal, and João Gama. Machine learning for streaming data: State of the art, challenges, and opportunities. 21(2):622, November 2019.

[132] Ioannis Kontopoulos, Antonios Makris, and Konstantinos Tserpes. A deep learn- ing streaming methodology for trajectory classification. *ISPRS International Journal of Geo-Information*, 10(4), 2021.

[133] A. Dosovitskiy and T. Brox. Inverting visual representations with convolutional networks. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4829–4837, 2016.

[134] Aravindh Mahendran and A. Vedaldi. Understanding deep image representations by inverting them. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5188–5196, 2015.

[135] L. Yang, J. Cao, H. Cheng, and Y. Ji. Multi-user computation partitioning for latency sensitive mobile cloud applications. *IEEE Transactions on Computers*, 64(8):2253–2266, August 2015.

[136] E. Dahlman, S. Parkvall, and J. Skold. *4G: LTE/LTE-Advanced for Mobile Broad- band*. Academic press, New York, 2013.

[137] J. Feng, L. Zhao, J. Du, X. Chu, and F. R. Yu. In *Energy-Efficient Resource Allocation in Fog Computing Supported IoT with Min-Max Fairness Guarantees*, Kansas City, MO, and May 2018. in Proc. *IEEE International Conference on Com- munications (ICC)*.

[138] X. Chen. Decentralized computation offloading game for mobile cloud computing. *IEEE Transactions on Parallel and Distributed Systems*, 26(4):974–983, April 2015.

[139] T. D. Burd and R. W. Brodersen. Processor design for portable systems. *Journal of VLSI signal processing systems for signal, image and video technology*, 13(2- 3):203–221, August 1996.

[140] S. Sardellitti, G. Scutari, and S. Barbarossa. Joint optimization of radio and computational resources for multicell mobile-edge computing. *IEEE Transactions on Signal and Information Processing over Networks*, 1(2):89–103, June 2015.

[141] Y. Mao, J. Zhang, SH. Song, and KB. Letaief. Power-delay tradeoff in multi-user mobile-edge computing systems. In *IEEE Global Communications Conference (GLOBECOM)*, Washington, DC, USA, December 2016. Proc.

[142] Z. Liang, Y. Liu, T. M. Lok, and K. Huang. Multiuser computation offloading and downloading for edge computing with virtualization. *IEEE Transactions on Wireless Communications*, 18(9):4298–4311, September 2019.

[143] F. Wang, J. Xu, X. Wang, and S. Cui. Joint offloading and computing optimiza- tion in wireless powered mobile-edge computing system. In *IEEE International Conference on Communications (ICC)*, Paris, France, May 2017. Proc.

[144] Y. Pochet and L. A. Wolsey. *Production Planning by Mixed Integer Programming*. Springer, Science & Business Media, 2006.

[145] P. Belotti, J. Lee, L. Liberti, F. Margot, and A. W"achter. Branching and bounds tightening techniques for non-convex minlp. *Optimization Methods & Software*, 24(4-5):597–634, August 2009.

[146] E. M. B. Smith and C. C. Pantelides. A symbolic reformulation/spatial branch- and-bound algorithm for the global optimisation of nonconvex minlps. *Computers & Chem. Eng*, 23:457–478, May 1999.

[147] K. Tammer. The application of parametric optimization and imbedding to the foundation and realization of a generalized primal decomposition approach. *Math- ematical research*, 35:376–386, 1987.

[148] R. Caruana. Multitask learning. *Machine Learning*, 28(1):41–75, July 1997.

[149] A. Argyriou, T. Evgeniou, and M. Pontil. Convex multi-task feature learning. *Machine Learning*, 73(3):243–272, 2008.

[150] S. Ruder. An overview of multi-task learning in deep neural networks. arXiv preprint, June 2017.

[151] F. N. Iandola, M. W. Moskewicz, K. Ashraf, and K. Keutzer. Firecaffe: near- linear acceleration of deep neural network training on compute clusters. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, NV, June 2016. Proc.

[152] M. Chen, U. Challita, W. Saad, C. Yin, and M. Debbah. *Artificial neural networks-based machine learning for wireless networks: A tutorial*. IEEE Com- munications

Surveys & Tutorials (Early Access, 2019.

[153] Z. X. Li, W. Wu, and Y. L. Tian. Convergence of an online gradient method for feedforward neural networks with stochastic inputs. *Journal of Computational and Applied Mathematics*, 163(1):165–176, 2004.

[154] S. Ruder. An overview of multi-task learning in deep neural networks. *arXiv preprint*, archivePrefix = arXiv, eprint = 1706.05098,, 2017.

[155] Y. Cao, T. Jiang, and C. Wang. Optimal radio resource allocation for mobile task offloading in cellular networks. *IEEE Network*, 28(5):68–73, September 2014.

[156] K. I. Ahmed, H. Tabassum, and E. Hossain. *Deep learning for radio resource allocation in multi-cell networks*. IEEE Network, Early Access, 2019.

[157] J. Nam, J. Kim, E. L. Mencía, I. Gurevych, and J. F"urnkranz. Large-scale multi-label text classification—revisiting neural networks. In *Proc. In Joint eu- ropean conference on machine learning and knowledge discovery in databases, , September*, 2014.

[158] D. M. Allen. Mean square error of prediction as a criterion for selecting variables. *Technometrics*, 13(3):469–475, 1971.

[159] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *International Conference for Learning Representations (ICLR)*, San Diego, May 2015. Proc.

[160] S. Guo, J. Liu, Y. Yang, B. Xiao, and Z. Li. Energy-efficient dynamic computation offloading and cooperative task scheduling in mobile cloud computing. *IEEE Transactions on Mobile Computing*, 18(2):319–333, 2019.

[161] J. B. Wang, J. Wang, Y. Wu, J. Y. Wang, H. Zhu, M. Lin, and J. Wang. A machine learning framework for resource allocation assisted by cloud computing. *IEEE Network*, 32(2):144–151, 2018.

[162] A. Shafer. *A Mathematical Theory of Evidence*. Princeton University Press, 1976.

[163] J. Dezert. Fondations pour une nouvelle theorie du raisonnement plausible et paradoxal. *ONERA Tech. Rep. RT 1/06769/DTIM*, 2003.

[164] F. Smarandache and J. Dezert. Advances and applications of dsmt for information fusion. *American Research Press, Rehoboth*, 2:3–68, 2006.

[165] Christopher D Manning, Christopher D Manning, and Hinrich Schütze. *Founda- tions of statistical natural language processing*. MIT press, 1999.

[166] Simon Tong and Daphne Koller. Support vector machine active learning with ap- plications to text classification. *Journal of machine learning research*, 2(Nov):45– 66, 2001.

[167] Juan Ramos et al. using tf-idf to determine word relevance in document queries. In *Proceedings of the first instructional conference on machine learning*, volume 242, pages 133–142. Piscataway, NJ, 2003.

[168] Jiaul H Paik. A novel tf-idf weighting scheme for effective ranking. In *Proceedings of the 36th international ACM SIGIR conference on Research and development in information retrieval*, pages 343–352. ACM, 2013.

[169] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[170] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[171] Rada Mihalcea and Carlo Strapparava. The lie detector: Explorations in the automatic recognition of deceptive language. In *Proceedings of the ACL-IJCNLP 2009 Conference Short Papers*, pages 309–312. Association for Computational Linguistics, 2009.

[172] James W Pennebaker, Martha E Francis, and Roger J Booth. Linguistic in- quiry and word count: Liwc 2001. *Mahway: Lawrence Erlbaum Associates*, 71(2001):2001, 2001.

[173] Ray Oshikawa, Jing Qian, and William Yang Wang. A survey on natural language processing for fake news detection. *arXiv preprint arXiv:1811.00770*, 2018.

[174] Natali Ruchansky, Sungyong Seo, and Yan Liu. Csi: A hybrid deep model for fake news detection. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, pages 797–806. ACM, 2017.

[175] Quoc Le and Tomas Mikolov. Distributed representations of sentences and documents. In *International conference on machine learning*, pages 1188–1196, 2014.

[176] Yunfei Long, Qin Lu, Rong Xiang, Minglei Li, and Chu-Ren Huang. Fake news detection through multi-perspective speaker profiles. In *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, volume 2, pages 252–256, 2017.

[177] Hamid Karimi, Proteek Roy, Sari Saba-Sadiya, and Jiliang Tang. Multi-source multi-class fake news detection. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 1546–1557, 2018.

[178] Jing Ma, Wei Gao, Zhongyu Wei, Yueming Lu, and Kam-Fai Wong. Detect rumors using time series of social context information on microblogging websites. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, pages 1751–1754. ACM, 2015.

[179] Tian Lan, Chen Li, and Jianxin Li. Mining semantic variation in time series for rumor detection via recurrent neural networks. In *2018 IEEE 20th International Conference on High Performance Computing and Communications; IEEE 16th International Conference on Smart City; IEEE 4th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, pages 282–289. IEEE, 2018.

[180] Takako Hashimoto, Tetsuji Kuboyama, and Yukari Shirota. Rumor analysis framework in social media. In *TENCON 2011-2011 IEEE Region 10 Conference*, pages 133–137.

IEEE, 2011.

[181] Olivier Chapelle, Bernhard Scholkopf, and Alexander Zien. Semi-supervised learn- ing (chapelle, o. et al., eds.; 2006)[book reviews]. *IEEE Transactions on Neural Networks*, 20(3):542–542, 2009.

[182] Xiaojin Jerry Zhu. Semi-supervised learning literature survey. Technical report, University of Wisconsin-Madison Department of Computer Sciences, 2005.

[183] Xiaojin Zhu and Zoubin Ghahramani. Learning from labeled and unlabeled data with label propagation. 2002.

[184] Martin Szummer and Tommi Jaakkola. Partially labeled classification with markov random walks. In *Advances in neural information processing systems*, pages 945–952, 2002.

[185] Michael Speriosu, Nikita Sudan, Sid Upadhyay, and Jason Baldridge. Twitter polarity classification with label propagation over lexical links and the follower graph. In *Proceedings of the First workshop on Unsupervised Learning in NLP*, pages 53–63, 2011.

[186] Gisel Bastidas Guacho, Sara Abdali, and Evangelos E Papalexakis. Semi- supervised content-based fake news detection using tensor embeddings and label propagation. In *Proc. SoCal NLP Symposium*, 2018.

[187] Olivier Chapelle and Alexander Zien. Semi-supervised classification by low den- sity separation. In *AISTATS*, volume 2005, pages 57–64. Citeseer, 2005.

[188] Kamal Nigam, Andrew McCallum, and Tom M Mitchell. Semi-supervised text classification using em.

[189] Lei Shi, Rada Mihalcea, and Mingjun Tian. Cross language text classification by model translation and semi-supervised learning. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 1057– 1067, 2010.

[190] Li Zhao, Minlie Huang, Ziyu Yao, Rongwei Su, Yingying Jiang, and Xiaoyan Zhu. Semi-

supervised multinomial naive bayes for text classification by leverag- ing word-level statistical constraint. In *Thirtieth AAAI conference on artificial intelligence*, 2016.

[191] Shusen Zhou, Qingcai Chen, and Xiaolong Wang. Fuzzy deep belief networks for semi-supervised sentiment classification. *Neurocomputing*, 131:312–322, 2014.

[192] Kamran Kowsari. *Investigation of fuzzyfind searching with golay code transforma- tions*. PhD thesis, M. Sc. Thesis, The George Washington University, Department of Computer Science, 2014.

[193] Kamran Kowsari, Maryam Yammahi, Nima Bari, Roman Vichr, Faisal Alsaby, and Simon Y Berkovich. Construction of fuzzyfind dictionary using golay coding transformation for searching applications. *arXiv preprint arXiv:1503.06483*, 2015.

[194] M. Basseville and I. Nikiforov. *Detection of Abrupt Change Theory and Application*. Prentice Hall., Englewood Cliffs, NJ, 1993.

[195] C. Hsu, D. Zhang, C. Yang, and H. Chu. An efficient method for optimizing reader deployment and energy saving. *Sensor Letters*, 11(9):1695–1703, 2013.

[196] T. Lin and H. Chang. Black hole traffic anomaly detections in wireless sensor network. *International Journal of Grid and High Performance Computing*, 7:42– 51, 2015.

[197] R. Lomotey and R. Deters. Unstructured data mining: use case for couchdb. *International Journal of Big Data Intelligence*, 2(3):168–182, 2015.

[198] G. Shafer. Perspectives on the theory and practice of belief functions. *International Journal of Approximate Reasoning*, 4(5):323–362, 1990.

[199] R. Yager and L. Liu. Classic works of the dempster-shafer theory of belief functions. *American Research Press, Rehoboth*, 2:3–68, 2006.

[200] B. Khaleghi, A. Khamis, O. Karray, and S. Razavi. Multisensor data fusion: A review

of the state-of-the-art. *Information Fusion*, 14(1):28–44, 2013.

[201] F. Smarandache and J. Dezert. Advances and applications of dsmt for information fusion. *American Research Press, Rehoboth*, 1, 2004.

[202] H. Jafari, X. Li, L. Qian, and Y. Chen. Community based sensing: A test bed for environment air quality monitoring using smartphone paired sensors. *36th IEEE Sarnoff Symposium*, pages 12–17, 2015.

[203] Gordon Pennycook and David G Rand. Fighting misinformation on social media using crowdsourced judgments of news source quality. *Proceedings of the National Academy of Sciences*, page 201806781, 2019.

[204] Kai Shu, Amy Sliva, Suhang Wang, Jiliang Tang, and Huan Liu. Fake news de- tection on social media: A data mining perspective. *ACM SIGKDD Explorations Newsletter*, 19(1):22–36, 2017.

[205] Hunt Allcott and Matthew Gentzkow. Social media and fake news in the 2016 election. *Journal of economic perspectives*, 31(2):211–36, 2017.

[206] Nir Grinberg, Kenneth Joseph, Lisa Friedland, Briony Swire-Thompson, and David Lazer. Fake news on twitter during the 2016 us presidential election. *Sci- ence*, 363(6425):374–378, 2019.

[207] Dirk Hovy. The enemy in your own camp: How well can we detect statistically-generated fake reviews–an adversarial study? In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Pa- pers)*, volume 2, pages 351–356, 2016.

[208] William Yang Wang. " liar, liar pants on fire": A new benchmark dataset for fake news detection. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, volume 2, pages 422–426, 2017.

[209] Martin Potthast, Johannes Kiesel, Kevin Reinartz, Janek Bevendorff, and Benno Stein. A stylometric inquiry into hyperpartisan and fake news. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Vol- ume 1: Long Papers)*, pages 231–240, 2018.

[210] Jing Ma, Wei Gao, Prasenjit Mitra, Sejeong Kwon, Bernard J Jansen, Kam-Fai Wong, and Meeyoung Cha. Detecting rumors from microblogs with recurrent neural networks. In *Ijcai*, pages 3818–3824, 2016.

[211] Zhe Zhao, Paul Resnick, and Qiaozhu Mei. Enquiring minds: Early detection of rumors in social media from enquiry posts. In *Proceedings of the 24th Inter- national Conference on World Wide Web*, pages 1395–1405. International World Wide Web Conferences Steering Committee, 2015.

[212] Sejeong Kwon, Meeyoung Cha, Kyomin Jung, Wei Chen, and Yajun Wang. Prominent features of rumor propagation in online social media. In *2013 IEEE 13th International Conference on Data Mining*, pages 1103–1108. IEEE, 2013.

[213] Yi Chang, Makoto Yamada, Antonio Ortega, and Yan Liu. Lifecycle modeling for buzz temporal pattern discovery. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 11(2):20, 2016.

[214] Yi Chang, Makoto Yamada, Antonio Ortega, and Yan Liu. Ups and downs in buzzes: Life cycle modeling for temporal pattern discovery. In *2014 IEEE International Conference on Data Mining*, pages 749–754. IEEE, 2014.

[215] Hannah Rashkin, Eunsol Choi, Jin Yea Jang, Svitlana Volkova, and Yejin Choi. Truth of varying shades: Analyzing language in fake news and political fact- checking. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 2931–2937, 2017.

[216] Cristina Bosco, Vincenzo Lombardo, Leonardo Lesmo, and Vassallo Daniela. Building a treebank for italian: a data-driven annotation schema. In *LREC 2000*, pages 99–105. ELDA, 2000.

[217] Yoon Kim. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*, 2014.

[218] Arkaitz Zubiaga, Maria Liakata, Rob Procter, Geraldine Wong Sak Hoi, and Peter Tolmie. Analysing how people orient to and spread rumours in social media by looking at conversational threads. *PloS one*, 11(3):e0150989, 2016.

[219] A. Dempster. Upper and lower probabilities induced by a multivalued mapping. *Annals of Mathematical Statistics*, 38:325–339, 1967.

[220] P. Smets. Constructing the pignistic probability function in a context of uncer- tainty. *Uncertainty in Artificial Intelligence*, 5:29–39, 2004.

[221] F. Smarandache and J. Dezert. Advances and applications of dsmt for information fusion. *American Research Press, Rehoboth*, 2:69–88, 2006.

[222] F. Smarandache and J. Dezert. An algorithm for quasi-associative and quasi- markovian rules of combination in information fusion. In *5th International Sym- posium on Applied Computational Intelligence and Informatics*, pages 557–562, 2004.

[223] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Dis- tributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[224] Yoav Goldberg and Omer Levy. word2vec explained: deriving mikolov et al.'s negative-sampling word-embedding method. *arXiv preprint arXiv:1402.3722*, 2014.

[225] Zhanpeng Zhang, Ping Luo, Chen Change Loy, and Xiaoou Tang. Facial landmark detection by deep multi-task learning. In *European conference on computer vision*, pages 94–

108. Springer, 2014.

[226] Sebastian Ruder. An overview of multi-task learning in deep neural networks. *arXiv preprint arXiv:1706.05098*, 2017.

[227] Shanta Chowdhury, Xishuang Dong, Lijun Qian, Xiangfang Li, Yi Guan, Jin- feng Yang, and Qiubin Yu. A multitask bi-directional rnn model for named entity recognition on chinese electronic medical records. *BMC bioinformatics*, 19(17):499, 2018.

[228] Xishuang Dong, Shanta Chowdhury, Lijun Qian, Xiangfang Li, Yi Guan, Jinfeng Yang, and Qiubin Yu. Deep learning for named entity recognition on chinese electronic medical records: Combining deep transfer learning with multitask bi- directional lstm rnn. *PloS one*, 14(5):e0216046, 2019.

[229] Samuli Laine and Timo Aila. Temporal ensembling for semi-supervised learning. *arXiv preprint arXiv:1610.02242*, 2016.

[230] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bot- tou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[231] Ross B. Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich fea- ture hierarchies for accurate object detection and semantic segmentation. *CoRR*, abs/1311.2524, 2013.

[232] J.R.R. Uijlings, K.E.A. van de Sande, T. Gevers, and A.W.M. Smeulders. Se- lective search for object recognition. *International Journal of Computer Vision*, 2013.

[233] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: To- wards real-time object detection with region proposal networks. In *Proceedings of the 28th International Conference on Neural Information Processing Systems- Volume 1*, NIPS'15, pages

91–99, Cambridge, MA, USA, 2015. MIT Press.

[234] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 779–788, 2016.

[235] Joseph Redmon and Ali Farhadi. YOLO9000: better, faster, stronger. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pages 6517–6525, 2017.

[236] Luca Bertinetto, Jack Valmadre, João F Henriques, Andrea Vedaldi, and Philip HS Torr. Fully-convolutional siamese networks for object tracking. *arXiv preprint arXiv:1606.09549*, 2016.

[237] Jack Valmadre, Luca Bertinetto, Joao Henriques, Andrea Vedaldi, and Philip H. S. Torr. End-to-end representation learning for correlation filter based tracking. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[238] B. Zhao, B. Xian, Y. Zhang, and X. Zhang. Nonlinear robust adaptive tracking control of a quadrotor uav via immersion and invariance methodology. *IEEE Transactions on Industrial Electronics*, 62(5):2891–2902, May 2015.

[239] Seungwon Choi, Suseong Kim, and H. Jin Kim. Inverse reinforcement learn- ing control for trajectory tracking of a multirotor uav. *International Journal of Control, Automation and Systems*, 15(4):1826–1834, Aug 2017.

[240] C. Kyrkou, G. Plastiras, T. Theocharides, S. I. Venieris, and C. Bouganis. Dronet: Efficient convolutional neural network detector for real-time uav applications. In *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 967–972, March 2018.

[241] George Plastiras, Christos Kyrkou, and Theocharis Theocharides. Efficient convnet-

based object detection for unmanned aerial vehicles by selective tile pro- cessing. In *Proceedings of the 12th International Conference on Distributed Smart Cameras*, ICDSC '18, pages 3:1–3:6, New York, NY, USA, 2018. ACM.

[242] Brais Bosquet, Manuel Mucientes, and Victor M. Brea. Stdnet: A convnet for small target detection. In *BMVC*, 2018.

[243] Alexander Wong, Mohammad Javad Shafiee, Francis Li, and Brendan Chwyl. Tiny SSD: A tiny single-shot detection deep convolutional neural network for real-time embedded object detection. *CoRR*, abs/1802.06488, 2018.

[244] P. Evdokiou D. Gouin and R. Vernik. A showcase of visualization approaches for military decision makers. http://ftp.rta.nato.int/public/PubFullText/RTO/MP/RTO-MP-105/MP-105-S2-01.pdf, 2002.

[245] Jaegul Choo and Haesun Park. Customizing computational methods for visual analytics with big data. *Computer Graphics and Applications, IEEE*, 33(4):22–28, July 2013.

[246] K. Reda, A Febretti, A Knoll, J. Aurisano, J. Leigh, A Johnson, M.E. Papka, and M. Hereld. Visualizing large, heterogeneous data in hybrid-reality environments. *Computer Graphics and Applications, IEEE*, 33(4):38–48, July 2013.

[247] Jinson Zhang and Mao Lin Huang. 5ws model for big data analysis and visu- alization. In *Computational Science and Engineering (CSE), 2013 IEEE 16th International Conference on*, pages 1021–1028, Dec 2013.

[248] S.J. Rysavy, D. Bromley, and V. Daggett. Dive: A graph-based visual-analytics framework for big data. *Computer Graphics and Applications, IEEE*, 34(2):26–37, Mar 2014.

[249] A Biem, H. Feng, AV. Riabov, and D.S. Turaga. Real-time analysis and man- agement of big time-series data. *IBM Journal of Research and Development*, 57(3/4):8:1–8:12, May 2013.

[250] Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. Speeding up convolutional neural networks with low rank expansions. In *Proceedings of the British Machine Vision Conference*. BMVA Press, 2014.

[251] https://developer.nvidia.com/tensorrt.

[252] J. Cheng, J. Wu, C. Leng, Y. Wang, and Q. Hu. Quantized cnn: A unified approach to accelerate and compress convolutional networks. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–14, 2018.

[253] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 4107–4115. Curran Associates, Inc., 2016.

[254] Sajid Anwar, Kyuyeon Hwang, and Wonyong Sung. Structured pruning of deep convolutional neural networks. *J. Emerg. Technol. Comput. Syst.*, 13(3):32:1– 32:18, February 2017.

[255] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Prun- ing convolutional neural networks for resource efficient transfer learning. *CoRR*, abs/1611.06440, 2016.

[256] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861, 2017.

[257] Mark Sandler, Andrew G. Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation. *CoRR*, abs/1801.04381, 2018.

[258] Yonggao Yang, J.X. Chen, and Woosung Kim. Gene expression clustering and 3d visualization. *Computing in Science Engineering*, 5(5):37–43, Sept 2003.

[259] Yonggao Yang, Changqian Zhu, and Hua Zhang. Real-time simulation: Water droplets on glass windows. *Computing in Science and Engg.*, 6(4):69–73, July 2004.

[260] Yonggao Yang, Jim X. Chen, and Mohsen Beheshti. Nonlinear perspective pro- jections and magic lenses: 3d view deformation. *IEEE Computer Graphics and Applications*, 25(1):76–84, 2005.

[261] Jian ao Lian and Yonggao Yang. A new cross subdivision scheme for surface design. *Journal of Mathematical Analysis and Applications*, 374(1):244 – 257, 2011.

[262] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 779–788, 2016.

[263] https://www.dac.com/content/2018-system-design-contest.

[264] C.-C. Jay Kuo. Understanding convolutional neural networks with A mathemat- ical model. *CoRR*, abs/1609.04112, 2016.

[265] http://cs231n.github.io/convolutional-networks/.

[266] Jan Hendrik Hosang, Rodrigo Benenson, and Bernt Schiele. Learning non- maximum suppression. *CoRR*, abs/1705.02950, 2017.

[267] Yuzhong Yan, Lei Huang, and Liqi Yi. Is apache spark scalable to seismic data analytics and computations? In *2015 IEEE International Conference on Big Data (Big Data)*, pages 2036–2045, 2015.

[268] Lei Huang and Yonggao Yang. Facilitating education using cloud computing infrastructure. *J. Comput. Sci. Coll.*, 28(4):19–25, April2013.

[269] Yuzhong Yan and Lei Huang. Large-scale Image Processing Research Cloud. In *CLOUD*

COMPUTING 2014, The Fifth International Conference on Cloud Computing, GRIDs, and Virtualization, Venice, Italy, May, 25-29 2014.

[270] Haoqiang Jin, Barbara Chapman, and Lei Huang. Performance Evaluation of a Multi-Zone Application in Different OpenMP Approaches. In *Proceedings of IWOMP 2007*.

[271] R.Xu. Openacc parallelization and optimization of nas parallel benchmarks, 2014.

[272] S.Chandrasekaran B.Chapman O. Hernandez R.Xu, C. Wang. A validation test- suite for openacc 1.0. In *IEEE 28th International Parallel and Distributed Pro- cessing Symposium Workshop PhD Forum (IPDPSW)*, 2014.

[273] Cheng Wang, Sunita Chandrasekaran, and Barbara Chapman. An openmp 3.1 validation testsuite. In *Proceedings of the 8th international conference on OpenMP in a Heterogeneous World*, IWOMP'12, pages 237–249, Berlin, Heidelberg, 2012. Springer-Verlag.

[274] The {OpenUH Compiler Project}. \url{http://www.cs.uh.edu/~openuh}, 2005.

[275] Lei Huang, Barbara Chapman, and Chunhua Liao. An Implementation and Evaluation of Thread Subteam for OpenMP Extensions. In *Programming Models for Ubiquitous Parallelism (PMUP 06)*, Seattle, WA, September 2006.

[276] Cody Addison, James LaGrone, Lei Huang, and Barbara Chapman. OpenMP 3.0 tasking implementation in OpenUH. In *Open64 Workshop at CGO 2009, In Con- junction with the International Symposium on Code Generation and Optimization (CGO)*, Boston, MA, April 2009.

[277] Lei Huang, Barbara Chapman, and Zhenying Liu. Towards a More Efficient Implementation of OpenMP for Clusters via Translation to Global Arrays. *Parallel Computing*, 31(10-12), 2005.

[278] Barbara M Chapman, Lei Huang, Gabriele Jost, Haoqiang Jin, and Bronis R de Su-

pinski. Support for Flexibility and User Control of Worksharing in OpenMP. Technical Report NAS-05-015, National Aeronautics and Space Administration, October 2005.

[279] Lei Huang, Deepak Eachempati, Marcus W Hervey, and Barbara Chapman. Ex- ploiting Global Optimizations for OpenMP Programs in the OpenUH Compiler. In *14th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPOPP 2009)*, Raleigh, NC, February 2009.

[280] Deepak Eachempati, Lei Huang, and Barbara M Chapman. Strategies and Implementation for Translating OpenMP Code for Clusters. In Ronald H Perrott, Barbara M Chapman, Jaspal Subhlok, Rodrigo Fernandes de Mello, and Laurence Tianruo Yang, editors, *HPCC*, volume 4782 of *Lecture Notes in Computer Science*, pages 420–431. Springer, 2007.

[281] Barbara M. Chapman, Babu Sundaram, and Kiran K. Thyagaraja. Ez-grid: Integrated resource brokerage services for computational grids (extended abstract), 2003.

[282] Barbara Chapman, Hari Donepudi, Yupeng Li, Priya Raghunath, Yonghong Yan, Babu Sundaram, and Jiwen He. Grid environment with web-based portal access for air quality modeling. In *Parallel And Distributed Scientific And Engineering Computing - 2004 Practice And Experience Advances In Computation: Theory And Practice*, volume 15, pages 191–208. NOVA Publishers, 2004.

[283] Barbara M Chapman, Hari Donepudi, Yupeng Li, Priya Raghunath, Babu Sundaram, Yonghong Yan, and Jiwen He. An ogsi-compliant portal for campus grids. In *ISPE CE*, pages 987–994, 2003.

[284] Yonghong Yan and Barbara M. Chapman. Scientific workflow scheduling in computational grids - planning, reservation, and data/network-awareness. In *8th IEEE/ACM International Conference on Grid Computing (GRID 2007), September 19-21, 2007,*

Austin, Texas, USA, Proceedings, pages 18–25. IEEE, 2007.

[285] Yonghong Yan and Barbara M. Chapman. Campus grids meet applications: Modeling, metascheduling and integration. *J. Grid Comput.*, 4(2):159–175, 2006.

[286] Y Yan and B M Chapman. Campus Grids Meet Applications: Modeling, Metascheduling and Integration. *Journal of Grid Computing*, 4(2):159–175, 2006.

[287] Barbara M Chapman, Priya Raghunath, Babu Sundaram, and Yonghong Yan. Predicting air quality in a production-quality grid environment. Technical report, Citeseer, 2005.

[288] Dongni Han, Shixiong Xu, Li Chen, and Lei Huang. Pads: A pattern-driven stencil compiler-based tool for reuse of optimizations on gpgpus. In *2011 IEEE 17th International Conference on Parallel and Distributed Systems*, pages 308– 315, 2011.

[289] Kaushik Datta, Mark Murphy, Vasily Volkov, Samuel Williams, Jonathan Carter, Leonid Oliker, David Patterson, John Shalf, and Katherine Yelick. Stencil computation optimization and auto-tuning on state-of-the-art multicore architectures. In *SC '08: Proceedings of the 2008 ACM/IEEE Conference on Supercomputing*, pages 1–12, 2008.

[290] Yuzhong Yan, Chao Chen, and Lei Huang. A productive cloud computing plat- form research for big data analytics. In *2015 IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 499–502, 2015.

[291] Alex Krizhevsky. Learning multiple layers of features from tiny images. *University of Toronto*, 05 2012.

[292] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[293] Francois Chollet. *Deep Learning with Python*. Manning Publications Co., USA, 1st edition, 2017.

[294] T. Q. Dinh, Q. D. La, T. Q. Quek, and H. Shin. Learning for computation offloading in mobile edge computing. *IEEE Transactions on Communications*, 66(12):6353–6367, August 2018.

[295] D. Zhang, J. Tang, W. Du, J. Ren, and G. Yu. Joint optimization of computation offloading and ul/dl resource allocation in mec systems. In *IEEE Annual International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)*, Bologna, Italy, September 2018. Proc.

[296] B. Yang, X. Cao, and L. Qian. A scalable mac framework for internet of things assisted by machine learning. In *Proc. IEEE VTC2018-Fall*, IL, August, 2018. Chicago.

[297] B. Yang, X. Cao, Z. Han, and L. Qian. A machine learning enabled mac framework for heterogeneous internet-of-things networks. *IEEE Transactions on Wireless Communications*, 18(7):3697–3712, July 2019.

[298] J. Hedengren. *MATLAB toolbox for the APMonitor Modeling Language*. [online], 2014.

[299] G. Pataki, M. Tural, and E. B. Wong. Basis reduction and the complexity of branch-and-bound. In *ACM-SIAM symposium on discrete algorithms*, Philadelphia, PA, January, 2010. Proc.

[300] H. Jafari, X. Li, and L. Qian. Efficient processing of uncertain data using dezert-smarandache theory: A case study. *2016 IEEE 14th Intl Conf on Dependable, Autonomic and Secure Computing, 14th Intl Conf on Pervasive Intelligence and Computing, 2nd Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress(DASC/PiCom/DataCom/CyberSciTech)*, pages 715–722, 2016.

[301] F. Smarandache and J. Dezert. Advances and applications of dsmt for information fusion. *American Research Press, Rehoboth*, 1:37–48, 2004.

[302] P. Djiknavorian and D. Grenier. Reducing dsmt hybrid rule complexity through op-timization of the calculation algorithm. *in: Advances and Application of DSmT for Information Fusion*, 2:365–429, 2006.

[303] Anand Rajaraman and Jeffrey David Ullman. *Mining of massive datasets*. Cam- bridge University Press, 2011.

[304] Elena Kochkina, Maria Liakata, and Arkaitz Zubiaga. All-in-one: Multi-task learning for rumour verification. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 3402–3413, 2018.

[305] Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level convolutional net- works for text classification. In *Advances in neural information processing sys- tems*, pages 649–657, 2015.

[306] Alexis Conneau, Holger Schwenk, Loïc Barrault, and Yann Lecun. Very deep convolutional networks for text classification. *arXiv preprint arXiv:1606.01781*, 2016.

[307] Peng Zhou, Wei Shi, Jun Tian, Zhenyu Qi, Bingchen Li, Hongwei Hao, and Bo Xu. Attention-based bidirectional long short-term memory networks for relation classification. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 207–212, 2016.

[308] Siwei Lai, Liheng Xu, Kang Liu, and Jun Zhao. Recurrent convolutional neural networks for text classification. In *Twenty-ninth AAAI conference on artificial intelligence*, 2015.

[309] Yiming Yang. A study of thresholding strategies for text categorization. In *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 137–145, 2001.

[310] Nitesh V Chawla. Data mining for imbalanced datasets: An overview. In *Data mining and knowledge discovery handbook*, pages 875–886. Springer, 2009.

APPENDIX A – PUBLICATIONS AND PRESENTATIONS (APRIL 2015 - MAY 2022)

Journal Papers

1) O. Fagbohungbe, S. Reza, X. Dong, L. Qian (2022). "Efficient Privacy Preserving Edge Intelligent Computing Framework for Image Classification in IoT," *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 6, no. 4, pp. 941-956, Aug. 2022, doi: 10.1109/TETCI.2021.3111636.

2) O. Onasami, M. Feng, H. Xu, M. Haile, L. Qian. (2022). "Underwater Acoustic Communication Channel Modeling using Reservoir Computing," *IEEE Access,* vol. 10, pp. 56550-56563, 2022, doi: 10.1109/ACCESS.2022.3177728.

3) B. Yang, X. Cao, C. Huang, C. Yuen, M. Renzo, Y. Guan, D. Niyato, L. Qian, and M. Debbah (2022). "Federated Spectrum Learning for Reconfigurable Intelligent Surfaces-Aided Wireless Edge Networks," *IEEE Transactions on Wireless Communications,* doi: 10.1109/TWC.2022.3178445.

4) X. Dong, S. Chowdhury, U. Victor, X. Li, L. Qian. (2022). "Semi-supervised Deep Learning for Cell Type Identification from Single-Cell Transcriptomic Data," *IEEE/ACM Transactions on Computational Biology and Bioinformatics,* doi: 10.1109/TCBB.2022.3173587.

5) O. Fagbohungbe, L. Qian (2022). "The Effect of Batch Normalization on Noise Resistant Property of Deep Learning Models," *IEEE Access.*

6) B. Yang, X. Cao, K. Xiong, C. Yuen, Y. Guan, S. Leng, L. Qian, and Z. Han (2021). "Edge Intelligence for Autonomous Driving in 6G Wireless System: Design Challenges and Solutions," *IEEE Wireless Communications Magazine,* vol. 28, no. 2, pp. 40-47, April 2021, doi: 10.1109/MWC.001.2000292.

7) B. Yang, X. Cao, C. Huang, C. Yuen, and L. Qian (2021). "Intelligent Spectrum Learning for Wireless Networks with Reconfigurable Intelligent Surfaces," *IEEE Transactions on Vehicular Technology,* vol. 70, no. 4, pp. 3920-3925, April 2021, doi: 10.1109/TVT.2021.3064042.

8) S.R. Reza, Y. Yan, X. Dong, and L. Qian (2021). "Inference Performance Comparison of Convolutional Neural Networks on Edge Devices." In: Paiva S., Lopes S.I., Zitouni R., Gupta N., Lopes S.F., Yonezawa T. (eds) *Science and Technologies for Smart Cities. SmartCity360° 2020*. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, vol 372. Springer, Cham. https://doi.org/10.1007/978-3-030-76063-2_23.

9) D. Adesina, C. Hsieh, Y. E. Sagduyu, and L. Qian (2021). "Adversarial Machine Learning in Wireless Communications using RF Data: A Review," submitted to *IEEE Communications Surveys & Tutorials, arXiv:2012.14392.*

10) J. Bassey, X. Li, and L. Qian (2021). "Device Authentication Codes based on RF Fingerprinting using Deep Learning," *EAI Endorsed Transactions on Security and Safety, arXiv:2004.08742.*

11) J. Bassey, X. Li, and L. Qian (2021). "A Survey of Complex-Valued Neural Networks," submitted to *The Proceedings of the IEEE, arXiv:2101.12249.*

12) X. Dong, U. Victor, and L. Qian (2021). "Semi-supervised Bidirectional RNN for Misinformation Detection," submitted to *Applied Intelligence.*

13) L. Nwuso, X. Li, L. Qian, S. Kim, and X. Dong (2021). "Semi-supervised Learning for COVID-19 Image Classification via ResNet," *EAI Endorsed Transactions on Bioengineering and Bioinformatics.*

14) B. Yang, X. Cao, C. Yuen and L. Qian (2021). "Offloading Optimization in Edge Computing for Deep-Learning-Enabled Target Tracking by Internet of UAVs," in *IEEE Internet of Things Journal*, vol. 8, no. 12, pp. 9878-9893, June 2021, doi: 10.1109/JIOT.2020.3016694.

15) Z. Zhou, and H. Xu (2021). "Decentralized Adaptive Optimal Tracking Control for Massive Autonomous Vehicle Systems with Heterogeneous Dynamics: A Stackelberg Game," *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

16) Z. Zhou, and H. Xu (2021). "Decentralized optimal large scale multi-player pursuit-evasion strategies: A Mean Field Game approach," *Neurocomputing*, vol. 484., pp: 46-58, 2021.

17) B. Yang, O. Fagbohungbe, X. Cao, C. Yuen, L. Qian, D. Niyato, and Y. Zhang (2021). "A Joint Energy and Latency Framework for Transfer Learning over 5G Industrial Edge Networks," *in IEEE Transactions on Industrial Informatics,* doi: 10.1109/TII.2021.3075444.

18) N. Kulathunga, N. Ranasinghe, D. Vrinceanu, Z. Kinsman, L. Huang, Y. Wang (2021). "Effects of Nonlinearity and Network Architecture on the Performance of Supervised Neural Networks," Section: Evolutionary Algorithms and Machine Learning, *Algorithms*.

19) X. Dong, U. Victor and L. Qian (2020). "Two-Path Deep Semisupervised Learning for Timely Fake News Detection," in *IEEE Transactions on Computational Social Systems*, vol. 7, no. 6, pp. 1386-1398, Dec. 2020, doi: 10.1109/TCSS.2020.3027639.

20) B. Yang, X. Cao, J. Bassey, X. Li, and L. Qian (2020). "Computation Offloading in Multi-Access Edge Computing Networks: A Multi-Task Learning Approach." in *IEEE Transactions on Mobile Computing,* doi: 10.1109/TMC.2020.2990630.

21) B. Yang, X. Cao, X. Li, C. Yuen, and L. Qian (2020). "Lessons Learned from Accident of Autonomous Vehicle Testing: An Edge Learning-aided Offloading Framework," *in IEEE Wireless Communications Letters,* vol. 9, no. 8, pp. 1182-1186, Aug. 2020, doi: 10.1109/LWC.2020.2984620.

22) X. Cao, Z. Song, B. Yang, L. Qian, Z. Han (2020). "Full-Duplex MAC in LAA/Wi-Fi Co-existence Networks: Design, Modeling and Analysis", *in IEEE Transactions on Wireless Communications,* vol. 19, no. 8, pp. 5531-5546, Aug. 2020, doi: 10.1109/TWC.2020.2994278.

23) D. Adesina, J. Bassey, and L. Qian (2020). "Robust Deep Radio Frequency Spectrum Learning for Future Wireless Communications Systems," in *IEEE Access*, vol. 8, pp. 148528-148540, 2020, doi: 10.1109/ACCESS.2020.3015939.

24) C. Kotteti, X. Dong, and L. Qian (2020). "Ensemble Deep Learning on Time-Series Representation of Tweets for Rumor Detection in Social Media," *Appl. Sci. 2020, 10, 7541, Progress in Artificial Intelligence*.

25) B. Yang, X. Cao, O. Omotere, X. Li, and L. Qian (2020). "Improving Medium Access Efficiency with Intelligent Spectrum Learning," *in IEEE Access,* vol. 8, pp. 94484-94498, 2020, doi: 10.1109/ACCESS.2020.2995398.

26) Y. Yang, R. Ren and P. M. Johnson (2020). "VetLink: A Livestock Disease-Management System," in *IEEE Potentials*, vol. 39, no. 2, pp. 28-34, doi: 10.1109/MPOT.2019.2941568.

27) B. Yang, X. Cao, X. Li, Q. Zhang, and L. Qian (2020). "Mobile Edge Computing based Hierarchical Machine Learning Tasks Distribution for Industrial Internet-of-Things." *IEEE Internet-of-Things Journal*, Vol. 7, No. 3, pp.2169-2180, March 2020.

28) S. O. Bamgbose, X. Li, and L. Qian (2020). "Neural Network Based Nonlinear Adaptive Controller Design for a Class of Bilinear System", *IET cognitive computation and systems,* Vol. 2 Iss. 1, pp. 1-11.

29) X. Cao, Z. Song, B. Yang, M. ElMossallamy, L. Qian, and Z. Han (2020). "A Distributed Ambient Backscatter MAC Protocol for Internet-of-Things Networks." *IEEE Internet-of-Things Journal*, Vol. 7, No. 2, pp.1488-1501, Feb 2020.

30) B. Yang, X. Cao, Z. Han, and L. Qian (2019). "A Machine Learning Enabled MAC Framework for Heterogeneous Internet-of-Things Networks." *IEEE Transactions on Wireless Communications*, vol. 18, no. 7, pp. 3697-3712, July 2019. (Selected as most 50 popular paper in July and August in 2019).

31) M. Feng, L. Qian, H. Xu (2019). "Multi-Autonomous Robot Enhanced Ad-hoc Network

under Uncertain and Vulnerable Environment," *IEICE Transactions*, Vol.E102-B, No.10, Oct. 2019.

32) B. Li, L. Qian, D. Qiao, S. Shao (2019). "MAC for the Next Generation Networks in Unlicensed Band", editorial, *Mobile Networks and Applications*.

33) X. Dong, S. Chowdhury, L. Qian, X. Li, Y. Guan, J. Yang, and Q. Yu (2019). "Deep learning for named entity recognition on Chinese electronic medical records: combining deep transfer learning with multitask bi-directional lstm rnn," *PLoS ONE* 14(5): e0216046.

34) A. M. DeGennaro and L. Huang (2019). "Synthetic Data Generation Using Generative Adversarial Networks for Seismic Inversion and Interpretation," *Journal of Interpretation*.

35) M. Bari, A. M. Malik, A. Qawasmeh, B. Chapman (2019). "Performance and Energy Impact of OpenMP Runtime Configurations on Power Constrained Systems", *Journal of Sustainable Computing, Informatics and Systems*.

36) S. O. Bamgbose, X. Li, and L. Qian (2019). "Trajectory tracking control optimization with neural network for autonomous vehicles," *Advances in Science, Technology and Engineering Systems Journal,* Vol. 4, No. 1, pp.217-224.

37) J. Wang, Y. Gong, L. Qian, R. Jäntti, M. Pan, Z. Han (2018). "Data-Driven Optimization Based Primary Users' Operational Privacy Preservation", *IEEE Transactions on Cognitive Communications and Networking*, Vol.4(2), pp.357-367.

38) S. Chowdhury, X. Dong, L. Qian, X. Li, Y. Guan, J. Yang, Q. Yu (2018). "A Multitask bi-directional RNN Model for Named Entity Recognition on Electronic Medical Records", *BMC Bioinformatics*.

39) M. Wolfe, S. Lee, J. Kim, X. Tiana, R. Xu, B. Chapman and S. Chandrasekaran (2018). "The OpenACC data model: Preliminary study on its major challenges and implementations", in *Parallel Computing*, Volume 78, 2018.

40) H. Jafari, X. Li, L. Qian, A. Aved, T. Kroecker (2018). "Efficient Processing of Big Uncertain Data from Multiple Sensors with High Order Multi-Hypothesis: An Evidence Theoretic Approach", *International Journal of Big Data Intelligence*, Vol.5(3), pp.177-190.

41) Y. Wang, H. Li, and L. Qian (2017). "Belief Propagation and Quickest Detection Based Cooperative Spectrum Sensing in Heterogeneous and Dynamic Environments", *IEEE Transactions on Wireless Communications*, Vol.16(11), pp.7446-7459.

42) L. Qian, J. Zhu, S. Zhang (2017). "Survey of Wireless Big Data", *Journal of Communications and Information Networks*, 2(1), pp.1-18.

43) L. Huang, X. Dong, T. Clee (2017). "A Scalable Deep Learning Platform For Identifying Geological Features from Seismic Attributes," *The Leading Edge*, Vol. 36 no. 3 pp. 249-256, Mar. 2017.

44) D. Zhou, Y. Yang, and H. Yan (2016). "A Smart Virtual Eye Mobile System for the Visually Impaired", *IEEE Potentials*, Nov, 2016.

45) H. Jafari, X. Li, L. Qian, A. Aved, T. Kroecker (2016). "Multisensor Change Detection based on Big Time-Series Data and Dempster-Shafer Theory", *Concurrency and Computation: Practice and Experience*, Sep 2016, pp.1-11.

46) L. Qian, Z. Han, Y. Chen, C. Xu, D. Kataria (2016). "Editorial: Smart Device Enabled Sensor Networks: Theory and Practice (SDES)", *International Journal of Distributed Sensor Networks,* Vol.12(8), Aug 2016.

47) J. Kamto, L. Qian, W. Li, and Z. Han (2015). "λ-Augmented Tree for Robust Data Collection in Advanced Metering Infrastructure", *International Journal of Distributed Sensor Networks*, Hindawi publishing corp., 2015.

48) Y. Yan, M. Hanifi, L. Yi, and L. Huang (2015). "Building a Productive Domain-Specific Cloud for Big Data Processing and Analytics Service", *Journal of Computer and Communications*, Vol. 3, Issue 5, pp. 107-117, 2015.

Book Chapters

1) L. Huang, T. Clee and N. Ranasinghe. (2021). "Scientific Machine Learning for Improved Seismic Simulation and Inversion," Book Chapter in *Data Analytics in Energy Resources Exploration*, Elsevier, ISBN: 9780128223086.

2) S. O. Bamgbose, X. Li, and L. Qian (2018). "Control of complex biological systems utilizing the neural network predictor," *Computational Intelligence and Optimization Methods for Control Engineering*. Springer.

Peer-reviewed Conference Papers

1) O. Fagbohungbe and L. Qian (2022). "L1 Batch Normalization and Noise Resistant Property of Deep Learning Models," *2022 International Joint Conference on Neural Networks (IJCNN)*, 2022.

2) Dong, X., Dukes, X. A., Littleton, J., Neville, T., Rollerson, C., and Quinney, A. L. (2022). "Object Detection on Raspberry Pi," *ASEE Gulf Southwest Annual Conference*, Prairie View, Texas.

3) K. Mensah-Bonsu, B. Yang, A. Eroglu, H. Xu, L. Qian (2022). "Equivalent Circuit Model for Varactor-Loaded Reconfigurable Intelligent Surfaces," *IEEE International Symposium on Antennas and Propagation and USNC-URSI Radio Science Meeting (IEEE AP-S/URSI 2022)*.

4) Z. Zhou and H. Xu (2022). "Mean Field Game based Decentralized Optimal Charging Control for Large Scale of Electric Vehicles," *IFAC 6th Intelligent Control and Automation Science (ICONS)*, 2022.

5) O. Adekanmbi, L. Huang (2022). "Performance Comparisons for Python Libraries in Parallel Computing and Physical Simulation." *American Society for Engineering Education (ASEE),* 2022.

6) L. Huang, D. Vrinceanu, Y. Wang, N. Kulathunga, N. Ranasinghe (2022). "Discovering Nonlinear Dynamics Through Scientific Machine Learning." In: Arai K. (eds) *Intelligent Systems and Applications*. Lecture Notes in Networks and Systems, vol 294. Springer, Cham.

7) G. Verma, S. Finviya, A. Malik, M. Emani and B. Chapman (2022). "Towards Neural Architecture-Aware Exploration of Compiler Optimizations in a Deep Learning {graph} Compiler". In *Proceedings of the 19th ACM International Conference on Computing Frontiers*, CFW, Turin, Italy.

8) W. Lu, B. Shan, E. Raut, J. Meng, M. Araya-Polo, J. Doerfert, A.M. Malik, and B. Chapman (2022). "Towards Efficient Remote OpenMP Offloading", in *International Workshop on OpenMP (IWOMP)*, September 27th–30th, 2022, Chattanooga, Tennessee, USA.

9) O. Onasami, D. Adesina, and L. Qian (2021). "Underwater Acoustic Communication Channel Modeling Using Deep Learning," *The Fifteenth International Conference on Underwater Networks and Systems (WUWNet 2021)*.

10) O. Fagbohungbe and L. Qian (2021). "Benchmarking Inference Performance of Deep Learning Models on Analog Devices," *2021 International Joint Conference on Neural Networks (IJCNN)*, pp. 1-9, doi: 10.1109/IJCNN52387.2021.9534143.

11) Y. Zhang and H. Xu (2021). "Learning based Decentralized Optimal Control for Large Scale Multi-agent System by using Neural Networks and Discrete-time Mean Field Game," *IEEE National Aerospace & Electronics Conference*, 2021.

12) Y. Zhang, L. Qian and H. Xu (2021). "Intelligent Joint Beamforming and Distributed Power Control for UAV-assisted Ultra-Dense Network: A Hierarchical Optimization Approach," *IEEE National Aerospace & Electronics Conference*, 2021, pp. 184-190, doi: 10.1109/NAECON49338.2021.9696309.

13) L. Huang, D. Vrinceanu, Y. Wang, N. Kulathunga, and N. Ranasinghe (2021). "Discovering Nonlinear Dynamics Through Scientific Machine Learning," *IntelliSys 2021*.

14) E. Raut, J. Anderson, M. Araya-Polo and J. Meng (2021). "Porting and Evaluation of a Distributed Tasks-driven Stencil-based Application," *Workshop on Programming Models and Applications for Multicores and Manycores (PMAM) 2021*.

15) X. Cao, B. Yang, H. Zhang, L. Qian, C. Yuen and Z. Han (2021). "Reconfigurable Intelligent Surface Assisted Internet-of-Things: MAC Design and Optimization," *2021 IEEE Wireless Communications and Networking Conference Workshops (WCNCW)*, pp. 1-6, doi: 10.1109/WCNCW49093.2021.9420040.

16) S. Reza, X. Dong, and L. Qian (2021). "Robust Face Mask Detection using Deep Learning on IoT Devices," *IEEE ICC Workshop on COVI-COM: Communication, IoT, and AI technologies to counter COVID-19*.

17) Z. Zhou, and H. Xu (2021). "Decentralized Optimal Multi-agent System Tracking Control Using Mean Field Games with Heterogeneous Agent," *2021 IEEE Conference on Control Technology and Application (CCTA)*.

18) Z. Zhou, and H. Xu (2021). "Decentralized Optimal Tracking Control for Large-scale Multi-Agent Systems under Complex Environment: A Constrained Mean Field Game with Reinforcement Learning Approach," *2021 IEEE Conference on Control Technology and Application (CCTA)*.

19) Z. Zhou, Yuzhu Zhang and H. Xu (2021). "Reinforcement Learning-based Decentralized Optimal Control for Large-Scale Multi-agent System by Using Neural Networks and Discrete-time Mean Field Games," *2021 IEEE International Joint Conference on Neural Networks (IJCNN)*.

20) E. Raut, J. Anderson, M. Araya-Polo and J. Meng (2021). "Evaluation of Distributed Tasks in Stencil-based Application on GPUs," *2021 IEEE/ACM 6th International Workshop on Extreme Scale Programming Models and Middleware (ESPM2)*, 2021, pp. 45-52, doi: 10.1109/ESPM254806.2021.00011.

21) G. Verma, M. Emani, C. Liao, P. Lin, T. Vanderbruggen, X. Shen, and B. Chapman (2021). "HPCFAIR: Enabling FAIR AI for HPC Applications", In *IEEE/ACM Workshop on Machine Learning in High-Performance Computing Environments (MLHPC)*, November 15, 2021, St. Louis, Missouri, USA.

22) C. Liao, P. Lin, G. Verma, T. Vanderbruggen, M. Emani, Z. Nan, and X. Shen (2021). "HPC Ontology: Towards a Unified Ontology for Managing Training Datasets and AI Models for High-Performance Computing", In *IEEE/ACM Workshop on Machine Learning in High-Performance Computing Environments (MLHPC)*, November 15, 2021, St. Louis, Missouri, USA.

23) S. Chowdhury, X. Dong, O. A. Solis, L. Qian and X. Li (2020). "Cell Type Identification from Single-Cell Transcriptomic Data via Gene Embedding," *2020 19th IEEE International Conference on Machine Learning and Applications (ICMLA)*, 2020, pp. 258-263, doi: 10.1109/ICMLA51294.2020.00050.

24) B. Williams, X. Dong and L. Qian (2020). "Data Driven Network Monitoring and Intrusion Detection using Machine Learning," *2020 Seventh International Conference on Social Networks Analysis, Management and Security (SNAMS)*, 2020, pp. 1-7, doi: 10.1109/SNAMS52053.2020.9336569.

25) U. Victor, X. Dong, X. Li, P. Obiomon and L. Qian (2020). "Effective COVID-19 Screening using Chest Radiography Images via Deep Learning," *2020 Fourth International Conference on Multimedia Computing, Networking and Applications (MCNA)*, 2020, pp. 126-130, doi: 10.1109/MCNA50957.2020.9264294.

26) S. Reza, Y. Yan, X. Dong, and L. Qian (2020). "Inference Performance Comparison of Convolutional Neural Networks on Edge Devices," *EAI Edge-IoT 2020*.

27) S. Tian, J. Doerfert, B. Chapman (2020). "Concurrent Execution of Deferred OpenMP Target Tasks with Hidden Helper Threads," in *33rd Workshop on Languages and Compilers for Parallel Computing (LCPC 2020)*, Oct. 14–16, 2020.

28) Z. Zhou, and H. Xu (2020). "Biologically Inspired Decentralized Adaptive Optimal Tracking Control for Large Scale Multi-Agent Systems with Input Constraints," *The 16th IEEE International Conference on Control & Automation (ICCA 2020)*, Oct, 2020, Sapporo, Hokkaido, Japan.

29) E. Raut, J. Meng, M. Araya-Polo, and B. Chapman (2020). "Evaluating Performance of OpenMP Tasks in a Seismic Stencil Application," in *International Workshop on OpenMP 2020*. Sep 22, 2020, Austin, USA.

30) L. Huang, E. Clee, and N. Ranasinghe (2020). "Improving Seismic Wave Simulation and Inversion Using Deep Learning." *The Smoky Mountains Computational Sciences & Engineering Conference 2020*, Aug. 26-28.

31) Z. Zhou, and H. Xu (2020). "Mean-Field Game and Decentralized Intelligent Adaptive Pursuit Evasion Strategy for Massive Multi-Agent Systems under Uncertain Environment," *American Control Conference (ACC 2020)*, Denver, CO, USA.

32) Z. Zhou, and H. Xu (2020). "Biomimetic Optimal Tracking Control using Mean-Field Games and Spiking Neural Networks," *IFAC 2020 World Congress (IFAC)*, Berlin, Germany.

33) C. Duan, X. Li, L. Qian (2020). "Switching Control of Genetic Regulatory Networks With Dwell-Time and Sampling," *IMECE2020*.

34) B. Yang, X. Cao, J. Bassey, X. Li, and et. al. (2019). "Computation Offloading in Multi-Access Edge Computing Networks: A Multi-Task Learning Approach", *IEEE International Conference on Communications (ICC 2019)*, May 20-24, Shanghai, China.

35) J. Bassey, X. Li, and L. Qian (2019). "An Experimental Study of Multi-Layer Multi-Valued Neural Network", *The 2nd International Conference on Data Intelligence and Security (ICDIS 2019)*, June 28-30.

36) J. Kemp, L. Huang, T. Clee (2019). "Full Waveform Inversion Performance Analysis on GPU Clusters," *2019 Rice Oil & Gas HPC Conference*.

37) L. Huang (2019). "Toward an Automated Deep Learning System for Global Seismic Monitoring," *the 2nd Annual Machine Learning in Solid Earth Geoscience*.

38) H. Wu, Z. Zhou, M. Feng, Y. Yan, H. Xu, and L. Qian (2019). "Real-time Single Object Detection on The UAV," *International Conference on Unmanned Aircraft Systems, ICUAS'19*, June 11-14, Atlanta, GA, USA.

39) B. Yang, H. Wu, X. Cao, X. Li, and et. al. (2019). "Intelli-Eye: An UAV Tracking System with Optimized Machine Learning Tasks Offloading," *IEEE International Conference on Computer Communications (INFOCOM) Workshop 2019*, Apr 29-May 2, Paris, France.

40) L. Li and B. Chapman (2019). "Compiler Assisted Hybrid Implicit and Explicit GPU Memory Management under Unified Address Space", *International Conference for High Performance Computing, Networking, Storage, and Analysis (SC'19)*, November 2019, Denver, CO.

41) S. Tian, J. Doerfert, B. Chapman (2019). "Asynchronous OpenMP Offloading on NVIDIA GPUs," *LLVM Dev Workshop*.

42) X. Cao, Z. Song, B. Yang, M. ElMossallamy, L. Qian, and Z. Han (2019). "A Distributed MAC Using Wi-Fi to Assist Sporadic Backscatter Communications," *IEEE International Conference on Computer Communications (INFOCOM) Workshop 2019*, Apr 29-May 2, Paris, France.

43) B. Yang, X. Cao, X. Li, and et. al. (2019). "Joint Communication and Computing Optimization for Hierarchical Machine Learning Task Distribution", *IEEE Symposium on Computers and Communications (ISCC 2019)*, June 30 - July 3, Barcelona, Spain.

44) J. Bassey, D. Adesina, X. Li, and et.al. (2019). "Intrusion Detection for IoT Devices based on RF Fingerprinting using Deep Learning", *The Fourth International Conference on Fog and Mobile Edge Computing (FMEC 2019)*, June 10-13, Rome, Italy.

45) N.R. Ranasinghe, L. Huang, T. Clee and J. Kemp (2019). "A machine learning approach to discriminate between explosions and earthquakes," *AGU 2019*.

46) I. Khatri, X. Dong, J. Attia and L. Qian (2019). "Short-term Load Forecasting on Smart Meter via Deep Learning," *51st North American Power Symposium (NAPS)*, October 13-15, Wichita, Kansas, USA.

47) X. Cao, Z. Song, B. Yang, X. Du, L. Qian, and Z. Han (2019). "Deep Reinforcement Learning MAC for Backscatter Communication Relying on Wi-Fi Architecture", *IEEE Global Communications Conference (Globecom)*, Dec 9-13, Waikoloa, HI, USA.

48) X. Dong, H. Wu, Y. Yan, and L. Qian (2019). "Hierarchical Transfer Convolutional Neural Networks for Image Classification", *IEEE International Conference on Big Data*, Dec 9-12, Los Angeles, CA.

49) C. Kotteti, X. Dong, and L. Qian (2019). "Rumor Detection on Time-Series of Tweets via Deep Learning", *IEEE Military Communication Conference (Milcom 2019)*, Nov 12-14, Norfolk, VA, USA.

50) D. Adesina, J. Bassey, and L. Qian (2019). "Practical Radio Frequency Learning for Future Wireless Communication Systems", *IEEE Military Communication Conference (Milcom 2019)*, Nov 12-14, Norfolk, VA.

51) D. Adesina, O. Adagunodo, X. Dong, and L. Qian (2019). "Aircraft Location Prediction Using Deep Learning", *IEEE Military Communication Conference (Milcom 2019)*, Nov 12-14, Norfolk, VA, USA.

52) Z. Zhou, L. Qian, and H. Xu (2019). "Intelligent Decentralized Dynamic Power Allocation in MANET at Tactical Edge based on Mean-Field Game Theory", *IEEE Military Communication Conference (Milcom 2019)*, Nov 12-14, Norfolk, VA, USA.

53) C. Kotteti, X. Dong, and L. Qian (2018). "Multiple Time-Series Data Analysis for Rumor Detection on Social Media", *IEEE International Conference on Big Data*, Dec 10-13, 2018, Seattle, WA, USA.

54) Y. Shi, L. Huang, X. Dong, T. Liu, J. Ning (2018). "Fully convolutional neural network's application on fault detection," in the *Proceedings of the 2018 American Geophysical Union (AGU) Fall Meeting*, Washington, D.C., Dec 10-14.

55) H. Jafari, O. Omotere, D. Adesina, H. Wu, L. Qian (2018). "IoT Devices Fingerprinting Using Deep Learning"*, IEEE Military Communication Conference (MILCOM),* October 29-31, 2018, Los Angeles, CA, USA.

56) M. Feng, L. Qian, H. Xu (2018). "Multi-Robot Enhanced MANET Intelligent Routing at Uncertain and Vulnerable Tactical Edge", *IEEE Military Communication Conference (MILCOM),* October 29-31, 2018, Los Angeles, CA, USA.

57) L. Huang (2018). "Generate Big Data to Enable Deep Learning for Seismic Inversion," *the 2018 Rice Data Science Conference*, Houston, TX, Oct. 14-15, 2018.

58) O. Omotere, J. Fuller, L. Qian, and Z. Han (2018). "Spectrum Occupancy Prediction in Co-existing Wireless Systems using Deep Learning", *IEEE 88th Vehicular Technology Conference* (*VTC 2018)*, August 27–30, 2018, Chicago, IL.

59) B. Yang, X. Cao, and L. Qian (2018). "A Scalable MAC Framework for Internet of Things Assisted by Machine Learning", *IEEE 88th Vehicular Technology Conference* (*VTC 2018)*, August 27–30, 2018, Chicago, IL.

60) C. Kotteti, X. Dong, N. Li and L. Qian (2018). "Fake News Detection Enhancement with Data Imputation", *4th Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress*, August 12-15, 2018, Athens, Greece.

61) L. Huang, M. Polanco, T. Clee (2018). "Initial Experiments on Improving Seismic Data Inversion with Deep Learning," *2018 New York Scientific Data Summit (NYSDS)*, Brookhaven National Laboratory, Upton, NY, Aug. 6-8, 2018.

62) J. Bassey, X. Li, L. Qian, A. Aved, T. Kroecker (2018). "Efficient Computing of Dempster-Shafer Theoretic Conditionals for Big Hard/Soft Data Fusion", *21st International Conference on Information Fusion (FUSION 2018)*, July 10-14, 2018, Cambridge, UK.

63) S. Bamgbose, X. Li, L. Qian (2018). "Neural Network Optimized Controller for Motion and Position Control in Autonomous Systems", *14th IEEE International Conference on Control & Automation (ICCA 2018)*, June 11-16, 2018, Anchorage, Alaska.

64) P. Johnson *et al*. (2018). "An Innovative New Approach to Animal Care," *2018 IEEE Global Humanitarian Technology Conference (GHTC)*, pp. 1-5, doi: 10.1109/GHTC.2018.8601912.

65) S. Chowdhury, X. Dong, L. Qian, X. Li, Y. Guan, J. Yang, Q. Yu (2018). "A Multitask bi-directional RNN Model for Named Entity Recognition on Electronic Medical Records", *International Conference on Intelligent Biology and Medicine (ICIBM 2018)*, Los Angeles, CA (**NSF Student Travel Award**).

66) M. Bari, L. Stoltzfus, P. Lin, C. Liao, M. Emani and B. Chapman (2018). "Is Data Placement Optimization Still Relevant On Newer GPUs?", In *Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS18)*.

67) J. Kemp and B. Chapman (2018). "Mapping OpenMP to a Distributed Tasking Runtime", In *International Workshop on OpenMP*, September 2018, Barcelona, Spain.

68) L. Li, H. Finkel, M. Kong and B. Chapman (2018). "Manage OpenMP GPU Data Environment Under Unified Address Space", In *International Workshop on OpenMP*, September 2018, Barcelona, Spain.

69) S. Milakovic, Z. Budimlic, H. Pritchard, A. Curtis, B. Chapman, and V. Sarkar (2018). "SHCOLL – a Standalone Implementation of OpenSHMEM-style Collectives API", In *OpenSHMEM Workshop*, August 2018, Baltimore, MD.

70) M. Grossman, H. Pritchard, A. Curtis, and V. Sarkar (2018). "HOOVER: Distributed, Flexible, and Scalable Streaming Graph Processing on OpenSHMEM", In *OpenSHMEM Workshop*, August 2018, Baltimore, MD.

71) L. Huang, D. Mistry, X. Dong (2018). "Apply Generative Adversarial Networks for Synthetic Seismic Data Generation," submitted *to Workshop on Data Mining for Geophysics and Geology (DMG2), SIAM Conference on Data Mining (SDM2018)*.

72) R. Sobayo, H.Wu, R. Ray and L. Qian (2018). "Integration of Convolutional Neural Network and Thermal Images into Soil Moisture Estimation", *International Conference on Data Intelligence and Security* (*ICDIS 2018)*, April 8-10, South Padre Island, USA.

73) I. Olakodana, Y. Wang, L. Qian (2017). "Advanced Data Processing for Communication-constrained Underwater Domain", *The Eleventh ACM International Conference on Underwater Networks and Systems (WUWNet 2017)*, Nov. 6-8, Halifax, NS, Canada.

74) S. Bamgbose, X. Li, L. Qian (2017). "Closed Loop Control of Blood Glucose Level with Neural Network Predictor for Diabetic Patients", *IEEE HealthCom*, Oct 12-15, 2017, Dalian, China.

75) D. Mistry, Y. Zhu, L. Huang (2017). "Scalable Intelligent Oilfield Streaming Data Analytics Platform," the *Fifth Digital Oilfield Summit Forum & International Academic Conference (DOSFIAC 2017).*

76) J. Wang, Y. Gong, L. Qian, R. Jäntti, M. Pan, Z. Han (2017). "Primary Users' Operational Privacy Preservation via Data-Driven Optimization", *IEEE Globecom*, Singapore (***Best Paper Award***).

77) O. Omotere, L. Qian, R. Jäntti, M. Pan, Z. Han (2017). "Big RF Data Assisted Cognitive Radio Network Coexistence in 3.5GHz Band", *the 26th International Conference on Computer Communications and Networks (ICCCN 2017)*, July 31- Aug 3, Vancouver, Canada.

78) H. Asaadi and B. Chapman (2017). "Comparative Study of Deep Learning Framework in HPC Environments", *New York Scientific Data Summit (NYSDS)*, Aug 6-9, 2017, New York, NY, USA.

79) C. Chen, Y. Yan, L. Huang, and L. Qian (2017). "Implementing a Distributed Volumetric Data Analytics Toolkit on Apache Spark", *New York Scientific Data Summit (NYSDS)*, Aug 6-9, New York, NY, USA.

80) H. Jafari, X. Li, L. Qian, A. Aved, T. Kroecker (2017). "Evidence Theory Enabled Quickest Change Detection Using Big Time-Series Data from Internet of Things", *19th International Conference on Data Mining, Big Data, Database and Data System*, June 15-16, Toronto, Canada.

81) X. Dong, L. Qian, and L. Huang (2017). "Short-Term Load Forecasting in Smart Grid: A Combined CNN and K-Means Clustering Approach", *IEEE International Conference on Big Data and Smart Computing (IEEE BigComp 2017)*, Juji, Korea.

82) O. Adejuwon, H. Wu, Y. Yan, and L. Qian (2017). "Performance Evaluation of Target Identification Model Using Deep Learning", *The 15th International Conference on Software Engineering Research and Practice*, July 17-20, Las Vegas, NV, USA.

83) X. Dong, L. Qian, and L. Huang (2017). "A CNN Based Bagging Learning Approach to Short-Term Load Forecasting in Smart Grid", *The 3rd IEEE International Conference on Cloud and Big Data Computing*, Aug 4 – 8, San Francisco, CA, USA.

84) J. Dennis, L. Huang, W. Lim, H. Wu, and Y. Yan (2017). "Implementing Deep Neural Networks on Fresh Breeze," *The International Parallel Computing Conference (Parco 2017),* Sep 12-15, Bologna, Italy.

85) L. Huang (2017). "Deep Learning on a GPU-enabled Cloud for Seismic Interpretation," *SEG Annual Conference* 2017, Houston, TX, Sep. 27, 2017.

86) M. Bari, A. Malik, A. Qawasmeh and B. Chapman (2017). "A Detailed Analysis of OpenMP Runtime Configurations for Power Constrained Systems", In *The Eighth International Green and Sustainable Computing Conference (IGSC2017*), October 2017, Orlando, FL (***Best Paper Award***).

87) X. Dong, S. Chowdhury, L. Qian, Y. Guan, J. Yang, Q. Yu (2017). "Transfer Bi-directional LSTM RNN for Named Entity Recognition in Chinese Electronic Medical Records", *IEEE HealthCom*, Oct 12-15, 2017.

88) L. Huang (2017). "Deep Learning Experiments for Seismic Interpretation," *Rice Data Science Conference* 2017, Houston, TX, Oct. 9-10, 2017.

89) H. Jafari, X. Li, L. Qian, A. Aved, T. Kroecker (2016). "Multisensor Change Detection based on Big Time-Series Data and Dempster-Shafer Theory", *IEEE International Conference on Big Data Science and Engineering (IEEE BigDataSE),* August 23-26, Tianjin, China.

90) H. Jafari, X. Li, L. Qian (2016). "Efficient Processing of Uncertain Data Using Dezert-Smarandache Theory: A Case Study", *IEEE International Conference on Big Data Intelligence and Computing (IEEE DataCom 2016)*, August 8-12, 2016, Auckland, New Zealand.

91) O. Omotere, W. Oduola, N. Zou, X. Li, L. Qian, and D. Kataria (2016). "Distributed Spectrum Monitoring and Surveillance using a Cognitive Radio based Testbed", *IEEE Sarnoff Symposium*, Sep 19-21, 2016, Newark, NJ, USA.

92) R. Xu, D. Khaldi, A. M. Malik, and B. Chapman (2016). "ACC-SVM: Accelerating SVM on GPUs using OpenACC", *The First Workshop of Mission-Critical Big Data Analytics,* Prairie View, TX, 2016.

93) A. M. Malik, L. Huang, and B. Chapman (2016). "Comparative Study of Distributed Machine Learning Frameworks using Spark", *The First Workshop of Mission-Critical Big Data Analytics,* Prairie View, TX, 2016.

94) Y. Yan, N. Del Rio, T. Lebo, L. Huang and L. Qian (2016). "Performance Evaluation of Big RDF Data on Cassandra", *The First Workshop of Mission-Critical Big Data Analytics,* Prairie View, TX, 2016.

95) X. Dong, L. Qian, Y. Guan, L. Huang, Q. Yu, J. Yang (2016). "Deep Learning based Ensemble Method for Named Entity Recognition in Electronic Medical Records", *2016 New York Scientific Data Summit.*

96) S. Bamgbose, Y. Zhang, and L. Qian (2016). "IMP-based Synchronization Controller for Distributed Three-Phase Inverter with Uncertain Loads", *IEEE International Conference on Mechatronics and Automation*, Aug 7-10, 2016.

97) R. Xu, S. Chandrasekaran, X. Tian, and B. Chapman (2016). "An Analytical Model-based Auto-tuning Framework for Locality-aware Loop Scheduling", *International Supercomputing Conference (ISC)*, Frankfurt, Germany, 2016.

98) C. Wang, S. Chandrasekaran, and B. Chapman (2016). "cusFFT: A High-Performance Sparse Fast Fourier Transform Algorithm on GPUs", *the 30th IEEE International Parallel & Distributed Processing Symposium (IPDPS)*, Chicago, Illinois, USA, 2016.

99) H. Asaadi, D. Khaldi and B. Chapman (2016). "A Comparative Survey of the HPC and Big Data Paradigms: Analysis and Experiments," *2016 IEEE International Conference on Cluster Computing (CLUSTER)*, Taipei, 2016, pp. 423-432.

100) C. Chen, Y. Yan , X. Dong and L. Huang (2016). "A Scalable and Productive Workflow-based Cloud Platform for Big Data Analytics," *2016 IEEE International Conference on Big Data Analysis* (ICBDA 2016), Hangzhou, China, March 12-14, 2016.

101) W. Oduola, N. Okafor, O. Omotere, L. Qian, and D. Kataria (2015). "Experimental Study of Hierarchical Software Defined Radio Controlled Wireless Sensor Network", in *Proceedings of the IEEE Sarnoff Symposium*, Newark, NJ, Sep 21-23, 2015.

102) H. Jafari, X. Li, L. Qian, and Y. Chen (2015). "Community Based Sensing: A Test Bed for Environment Air Quality Monitoring using Smartphone paired Sensors", in *Proceedings of the IEEE Sarnoff Symposium*, Newark, NJ, Sep 21-23, 2015.

103) Y. Yan, L. Huang, and L. Yi (2015). "Is Apache Spark Scalable to Seismic Data Analytics and Computations", *IEEE International Conference on Big Data*, Santa Clara, CA, Oct 29 – Nov 1, 2015.

104) Y. Yan, C. Chen, and L. Huang (2015). "A Productive Cloud Computing Platform Research for Big Data Analytics", *7th IEEE International Conference on Cloud Computing technology and Science* (*IEEE CloudCom 2015*), Vancouver, Canada, Nov 30 – Dec 3, 2015.

LIST OF SYMBOLS, ABBREVIATIONS, AND ACRONYMS

AI	Artificial Intelligence
AP	Access Point
API	Application Programming Interface
AWGN	Additive White Gaussian Noise
BER	Bit Error Rate
BP	Belief Propagation
BPSK	Binary Phase Shift Keying
BS	Base Station
CAP	Computational Access Point
CG	Code Generator
CNN	Convolutional Neural Network
CO	Carbon Monoxide
COTS	Commercial Off-The-Shelf
CREDIT	The Center of Excellence in Research and Education for Big Military Data Intelligence
DL	Deep Learning
DMAT	Distributed Multi-dimensional Array Toolkit
DNN	Deep Neural Networks
DoD	Department of Defense
DRF	Dominant Resource Fairness
DST	Dempster-Shafer Theory
DSTS	Dynamic Series-Time Structure
DSmT	Dezert-Smarandache Theory
ECC	Elliptic Curve Cryptosystem
ETSI	European Telecommunications Standards Institute
FFT	Fast Fourier Transform
GA	Global Array
GACA	Genetic Algorithm based Computation Algorithm
GPU	Graphics Processing Unit

GRU	Gated Recurrent Unit
HARNN	Hierarchical Attention RNN
HDFS	Hadoop Distributed File System
HPC	High Performance Computing
IBC	ID-based cryptography
IoT	Internet of Things
JTORA	Joint Task O_oading and Resource Allocation
LDM	Large Data Management
LIWC	Linguistic Inquiry and Word Count
LOD	Level of Detail
LOED	Leave-One-Event-Out
LSTM	Long Short-Term Memory
MCC	Mobile Cloud Computing
MDP	Markov Decision Process
MEC	Mobile Edge Computing
MES	Mobile Edge Computing Server
MINLP	Mixed Integer NonLinear Programming
ML	Machine Learning
MSE	Mean Square Error
MTFNN	Multi-Task learning based Feedforward Neural Network
MTL	Multi-Task learning
MU	Mobile User
NLP	Natural Language Processing
NPB	NASA Parallel Benchmark
OFDMA	Orthogonal Frequency Division Multiple Access
PAAS	Platform as a Service
PAC	Probably Approximately Correct
PCA	Principal Components Analysis
PSO	Particle Swarm Optimization

PUF	Physically Unclonable Functions
PVAMU	Prairie View A&M University
QoS	Quality-of-Service
RDD	Resilient Distributed Dataset
RFID	Radio Frequency IDenti_cation
RLTBB	Reformulation Linearization Technique-based Branch-and-Bound
RCNN	Recurrent CNN
RNN	Recurrent Neural Network
SAC	Seismic Analytics Cloud
SBB	Spatial Branch and Bound
SBT	Simple Build Tool
SEG	Society of Exploration Geophysicists
SGD	Stochastic Gradient Descent
SINR	Signal-to-Interference-plus-Noise Ratio
SOM	Self Organizing Map
SQL	Structured Query Language
TDSL	Two-path Deep Semi-supervised Learning
TF-IDF	Term Frequency-Inverse Document Frequency
THECB	Texas Higher Education Coordinating Board
TPU	Tensor Processing Unit
UAV	Unmanned Aerial Vehicle
VM	Virtual Machine
WLAN	Wireless Local Area Network
WSN	Wireless Sensor Network